SERIAL NO. S/N 102496 Model III

MMSFORTH™

USERS MANUAL

by A. Richard Miller

For specific use with the MMSFORTH Disk System (V2.0 or V2.1)
on Radio Shack TRS-80 Model I or III or IBM Personal Computer

(requires 1 minidisk drive and 32K RAM)

MMS MILLER MICROCOMPUTER SERVICES
61 LAKE SHORE ROAD, NATICK, MASS. 01760
(617) 653-6136

CREDITS:

Forth was conceived about 1970 by Charles Moore. The fundamental and continuing work by him and Elizabeth Rather, both now with Forth, Inc., is valued immensely by all who appreciate Forth programming.

The principal author of the MMSFORTH System is Tom Dowling. Tom has been building Forth systems since 1975 and has been perfecting MMSFORTH since 1978. MMS first released MMSFORTH as Version 1.5 in 1979 and introduced minor revisions through Version 1.9 while applying it to many commercial tasks.

Version 2.0, the first major revision of MMSFORTH, introduced the 79-STANDARD subset of Forth words and was released in June 1981. It is the product of Tom Dowling, assisted by Dick and Jill Miller and John Rible.

Thanks are due to the many of our two thousand prior users who suggested new features and reported sharp edges. We particularly thank the MMSFORTH Users Group of Eastern Massachusetts and the users across the world who so effectively shake out our draft versions of MMSFORTH.

Third edition, July 1982
9 8 7 6 5 4 3 2 1 : Printing

This book is regularly updated as a component of the MMSFORTH System. Some chapters have been derived in part from "The microFORTH PRIMER", through a cooperative agreement with FORTH, INC. Text copy was produced with a TRS-80 Model III microcomputer, a NEC Spinwriter printer, and the FORTHWRITE wordprocessing system in MMSFORTH.

Miller Microcomputer Services
61 Lake Shore Road, Natick MA 01760-2099
(617) 653-6136

MMS MILLER MICROCOMPUTER SERVICES
61 LAKE SHORE ROAD, NATICK, MASS. 01760
(617) 653-6136

July, 1982

TO PURCHASERS OF MMSFORTH:

Thank you for your order.

MMS is very serious about supporting MMSFORTH with major applications programming and with information and upgrades as opportunities arise. In order to provide these services, we need your name and your agreement not to distribute copies of this software to nonpurchasers. **Do not fail to complete and return the yellow copy of the MMSFORTH User License Agreement & Registration Form.** Upon its receipt we will send you the missing Glossary sections and you will be eligible for upgrade information, subscription to the MMSFORTH NEWSLETTER, etc.

We do not guarantee to answer all your questions about MMSFORTH without charging consulting fees. But we try to support reasonable requests, to encourage knowledgeable dealers, and to nourish a Newsletter which will share questions and answers as surely as MMSFORTH users send us their ideas. Above all, we urge you to provide us with good documentation of bugs, fixes, and improvements. Together we can continue to provide a remarkable microcomputer software environment!

Sincerely,

Dick Miller

Dick & Jill Miller/FORTHWRITE

Sign, then type or neatly print **ALL** requested information on the **YELLOW** copy of this form and **return within 14 days of purchase.** It is your guarantee that MMS will honor you as a bona fide user of MMSFORTH, and it will be your ticket to additional instructions and update mailings on the MMSFORTH System.

========== **MMSFORTH SYSTEM USER LICENSE AGREEMENT & REGISTRATION FORM** ========

This copy of the MMSFORTH System is licensed for use on a single computer (designated by its Serial Number) with support to the Designated Person whose signature appears below. Copies may be made for back-up purposes and modified versions may be generated for use on the same computer. Modified versions also must display the MMS copyright and MMS serial number information upon start-up. **Under no circumstances shall this agreement be construed as permission to distribute any original or modified version of MMSFORTH for sale, trade, as a gift or otherwise, except when ALL originals, copies and accessory MMSFORTH software are transferred to another computer and/or Designated Person via a separately negotiated MMS License Transfer.** MMS liability is limited to the provision of a machine-readable version of MMSFORTH.

Development and marketing of commercial products are encouraged provided that the MMSFORTH System is not included in whole or in part, except by separate sale or through a separate licensing agreement. Contact MMS for such arrangements.

Designated Person, Signature _____ Date ___/___/___

Last name _____ First (in full) _____

Company _____

Address _____

City, State, Zip _____

Phone Number (_____)_____ May we give your name to other users (Y/N)?_____

Licensed Computer: Make _____ Model _____ Serial Number _____

RAM Size _____K_ No. of Disk Drives _____

MMSFORTH System: Version ____ Serial Number (first screen) _____ Date Purchased ___/___/___

Bought from _____

Address _____

Comments (Interests and applications, suggestions - Use reverse side if desired.):

========== Fold this form, stamp and mail to: ==========

======
PLACE
STAMP
HERE!
======

MMSFORTH System Registration
MILLER MICROCOMPUTER SERVICES
61 Lake Shore Road
Natick, MA 01760
UNITED STATES OF AMERICA

MMS MILLER MICROCOMPUTER SERVICES
61 LAKE SHORE ROAD, NATICK, MASS. 01760
(617) 653-6136

========== PART II OF MMSFORTH SYSTEM INSTRUCTIONS, PLUS USER COMMENT FORM ==========

Thank you for properly registering your new MMSFORTH SYSTEM. This remaining part of the MMSFORTH SYSTEM instruction set supplements your initially supplied information. Merge the new pages and any replacement pages into place.

MMS hopes you will find the MMSFORTH SYSTEM and instructions among your most useful software acquisitions. To assist us in reaching that goal, we invite you to keep simple notes on the "rough edges" you experience while learning to use this package. MMS will appreciate your notes on the form below or in equivalent format, to help us to make MMSFORTH and your future purchases from MMS still better.

COMMENTS - TO BE FORWARDED TO MMS

MMSFORTH SYSTEM INSTRUCTIONS:
(Where possible, cite page number, specific phrase, nature of problem, suggested rewording, etc.)

MMSFORTH SYSTEM PERFORMANCE:
(Did you find an operation improper or undesirable? Suggest changes where appropriate.)

MARKETING PERFORMANCE:
(Are MMS and your dealer treating you right? Value for dollar, prompt delivery, condition of merchandise, availability of reasonable support, etc.)

OTHER SUGGESTIONS:

(Use reverse side or additional pages as needed.)

========== FOLD THIS FORM, STAMP AND MAIL TO: ==========

User's name:
MMSFORTH System S/N:

=====
PLACE
STAMP
HERE!
=====

MMSFORTH SYSTEM COMMENTS
MILLER MICROCOMPUTER SERVICES
61 LAKE SHORE ROAD
NATICK, MASS. 01760

TABLE OF CONTENTS

Tables, Listings & Figures ____ ii
Preface ____ P-1
Basic FORTH Operations ____ 1-1
Editing FORTH Blocks ____ 2-1
  Editing Commands ____ 2-5
  Editing Conventions ____ 2-12
Copying, Loading and Printing ____ 3-1
Data Declarations ____ 4-1
Handling Text ____ 5-1
Conditional Branches and Loops ____ 6-1
Program Development ____ 7-1
  PAINT ____ 7-3
  SIMPLE SIMON ____ 7-7
Advanced Program Development (CHECKBOOK) ____ 8-1

APPENDICES *

Getting Started ____ A1-1
Cassette Information (not included) ____ A2-1
MMSFORTH System Demo Programs ____ A3-1
MMSFORTH System Utilities ____ A4-1
  Translation from MMSFORTH V1.9 ____ A4-10
MMSFORTH System Extensions ____ A5-1
TROUBLESHOOTING in MMSFORTH ____ A6-1
  Error Messages ____ A6-1
MMSFORTH Memory Map ____ ** A7-1
CATALOG Listing of MMSFORTH Wordnames ____ ** A8-1
FORTH Glossary ____ ** A9-1
ASSEMBLER ____ ** A10-1
Keyboard Key Definitions ____ ** A11-1
System Constants ____ ** A12-1
Forth Programming Rules ____ A13-1
Bibliography ____ A14-1

* - No Appendices are provided when a USERS MANUAL is sold without the MMSFORTH System; refer to the FORTH-79 STANDARD MANUAL and/or the specific manual for your other Forth system.

** - Appendices A7-A12 are provided upon return of your properly completed MMSFORTH License Agreement.

TABLES

1. Arithmetic Operators _____ 1-14
2. Comparison Operators _____ 1-15
3. Stack Manipulation Operators _____ 1-16
4. Editing Commands _____ 2-11
5. Editing Conventions _____ 2-12
6. Memory Operators _____ 4-8
7. MMSFORTH System Indexes _____ A1-11
8. V1.9 to V2.0 Translation Tables _____ A4-13
9. Double-Precision Arithmetic Operations _____ A5-3
10. Keyboard Key Definitions _____ A11-1
11. System Constants _____ A12-1
12. Forth Programming Rules _____ A13-1

LISTINGS

1. STRINGS Example _____ 5-3
2. PAINT _____ 7-3
3. SIMPLE SIMON _____ 7-8
4. CHECKBOOK _____ 8-2

FIGURES

1. MMSFORTH Memory Map _____ A7-1
2. CATALOG Listing of MMSFORTH wordnames _____ A8-3

PREFACE

LICENSING

Forth is a remarkable and different way to see and use a computer system, and MMSFORTH is an original and professional version for the Radio Shack TRS-80 Models I, III and the IBM Personal Computer. In order to keep MMSFORTH powerful, well-supported and readily available to new users, Miller Microcomputer Services has chosen to market the standard versions of it inexpensively under single-computer licensing with one-person support. This means that you may copy and modify your singly licensed and serialized MMSFORTH system for your own personal use (but must maintain its front-screen copyright notice and serial number) and others may use your system copies on your one computer. However, **YOU MAY NOT DISTRIBUTE THE ORIGINAL OR COPIES OF YOUR MMSFORTH SYSTEM TO OTHERS FOR MONEY OR OTHERWISE.**

Organizations using MMSFORTH on more than one computer or with more than one supported person must buy an individual copy for each computer and/or supported user. Discount prices are available when at least five copies are ordered at once. Alternatively, a single license may be extended to all users within a single corporate site address, for a $1,000 surcharge. Additional Corporate Site License Extensions (CSLE's) cost $500 or less, and are required for additional sites or for extended use of certain programs such as FORTHWRITE, GENERAL LEDGER, AND TRADESHOW.

Any unauthorized exception to these requirements is software piracy. It discourages general support by good authors and publishers, it will result in loss of your specific support from MMS, and will leave you subject to legal prosecution. MMS requests that you abide by these requirements yourself, and that you impress others with the need for this level of responsibility in dealings with other computer software products. Where questions exist, please **contact MMS for clarification.**

Be sure to return your MMSFORTH Registration Form and License Agreement to MMS promptly, answered legibly and completely, with your full name (not just first initials!), signature, etc. to assure prompt return of the additional Appendix sections of your MMSFORTH Users Manuals.

SUPPORT SERVICES

Licensed MMSFORTH users may call MMS Monday through Friday between 9 a.m. and 9 p.m. Eastern U.S. Time for brief suggestions about troubleshooting. Busy lines usually preclude extensive conversation except for clients of our consulting services. **STATE YOUR MMSFORTH VERSION AND SERIAL NUMBER WHEN PLACING A CALL OR ORDER.**

MMS offers a variety of applications and utilities programs running in MMSFORTH. These include THE DATAHANDLER, a UTILITIES Diskette, a GAMES Diskette, the FORTHCOM communications program, the FORTHWRITE word processor, a GENERAL LEDGER and others. Each is individually documented on the source blocks and in accompanying literature.

MMS also supports its users with professional workshops and consulting, with volunteer MMSFORTH User Groups across the world, and with the MMSFORTH NEWSLETTER. Several issues of this periodical are advanced with the MMSFORTH Glossary; **subscribe** to assure notification of changes, fixes, and many useful and enjoyable program routines, explanations, etc.

USER INPUT AND UPGRADES

MMS solicits your feedback to improve our products and services. To this end, we include a User Comment Form with each of our major products.

We invite our licensed users to share in our continuing development efforts. Inexpensive rewrites of your original Forth and applications programs to our latest versions are available, as are new manuals when appropriate. Consult your MMSFORTH Newsletter for announcements in this regard. Poorly packaged diskettes often arrive damaged at MMS. To save our time and a $4.00 replacement fee, protect your disk from dust, moisture, bending and pressure, as follows: place your diskette in its protective envelope, then put both into a "sandwich baggie" and tape it shut. Nest this between two wads of crumpled paper, etc., inside a box (**never** an envelope!) and seal it so it is rigid. Then mail it to MMS.

ABOUT MMSFORTH

If you haven't yet used Forth, be prepared for a considerable difference and many surprises! Forth is a total computer software environment, requiring no other operating system or language. Forth seems to be all things to all people. Some of your early experiences will seem confusing because of this multiplicity of riches, but you will come to understand and appreciate the difference between Forth and your other computer environments.

Depending on how you use it, your Forth can be a simple way to talk to a computer in an English-like high-level language, or a very complex combination of the three traditional levels of computer languages: machine code, assembler, and high-level. A simple tool for the creation of new instructions or even of new words to create new **types** of instructions in the highest level ever available - the fourth level from which it takes its name. A way to combine language and operating system, or to combine the best features of interpreter and compiler.

Forth also is a way to speed up program development time (once you learn its fundamentals), and the program's actual run time speed. (Expect 10 to 20 times speed-up over your interpreter BASIC, nearly assembly language speed - with an assembler aboard in case you need faster!) A way to write a program in a small amount of RAM, even smaller than assembler programs in many cases. A flexible and powerful system suited to the creation of business, games, or process control programs, or for running a growing number of prepackaged Forth software applications. A method for automatically organizing your programming effort, with as little or as much built-in documentation as you choose. To many Forth programmers Forth also is a philosophy, and to some a religion. Prepare yourself for an exciting new experience in a new type of computer environment which will seem unbounded!

MMSFORTH is an original and unusually complete Forth system. Like all other versions of Forth, it owes its basic concepts to the pioneering work of Charles Moore, Elizabeth Rather, and others in the radio-astronomy community. It is not figForth or any of the Forth products of Forth, Inc.; however, MMS and these other Forth producers have cooperated to develop **79-STANDARD FORTH**, a subset of words which is incorporated within MMSFORTH V2.0 and other contemporary Forth systems.

This book is primarily designed to introduce new users to the standard MMSFORTH System Diskette. A MMSFORTH System Cassette is available on custom order and includes most of the same words. However, the latter does not support directory operations and will necessarily be somewhat more difficult to use, especially during the learning period. At MMS, we utilize it to run inexpensive satellite systems in distributed computer applications. Its appeal to the hobbyist must be balanced against

the lesser support it can demand in our competitive market. Custom MMSFORTH Systems are also available for hard disk and other hardware combinations.

Although this MMSFORTH USERS MANUAL is written for novice Forth programmers, we do assume a basic understanding of computer terms and concepts. Most new MMSFORTH users already can program their computer in BASIC. If you lack background, we suggest that you read any of the large number of books and magazines on small computer theory, register for a microcomputer computer course, or join one of the many popular computer clubs. The most important ingredients in your approach will be interest, a willingness to experiment with new concepts, and above all, the use of Forth on a live keyboard.

DIFFERENT STROKES FOR DIFFERENT KEYBOARDS

Readers using computers other than the Radio Shack TRS-80 Models I or III may be confused by some of the references to their keyboard layouts. The TRS-80's Enter key corresponds to Carriage Return or Return on other keyboards, while its Backarrow key corresponds to Backspace or Rubout. And in MMSFORTH we create other key assignments: Clear for a Control key, Shift-Clear for an Alternate key, Uparrow for Escape, etc.

Similarly, readers not using the IBM Personal Computer may wish to know that its keyboard provides Control and Alternate keys, as well as Delete, Insert, Home, End, PageUp, PageDn, PrintScreen and other keyboard options. In general, MMSFORTH preserves their original operation (i.e., Control-Break).

Although the IBM PC keyboard appears to provide **four** leftarrow keys, they are treated in differing and logical manners. The one on the left of the keyboard is the "Tab-left", a shifted Tab key. At the right, the top-most left-arrow is the "Back-arrow". The one beneath it is the Enter key, and the one below that (on the numeric keypad) is the Editor cursor's "Left-arrow". Have faith, you **will** learn!

USING THIS MANUAL

Tables have been included in this manual at the ends of Chapters 1, 2, and 3 for ease of later reference. There are a few fundamental procedures that must be observed in order to write clear Forth programs; we call these Rules. Each is set off in upper-case and numbered on its first appearance. They are also printed together as a list which makes up Appendix A13. Other Appendices provide information for beginning MMSFORTH users, unsupported internal information for experts, and general background material. Appendix A9 consists of a MMSFORTH Glossary that contains all commonly used words. It is provided in exchange for your properly completed MMSFORTH License Agreement, and is not available to non-users.

To make reading this documentation as easy as possible, the following conventions will be used:

1. Examine this manual's type font carefully to learn the differences between our printed O (the capital letter) and 0 (the number), between l (the lower-case letter) and 1 (the number), etc.

2. FORTH words that appear in prose passages as examples of commands are printed in capital letters, and are enhanced when first introduced. Words defined as occasional examples are also set off.

3. Where there might be confusion about who types what, the **computer's** output is underlined.

4. In all examples that show stack usage, the top item of the stack appears to the right (as it does on the video screen when you are entering).

5. Brief examples of definitions are provided as often as possible. After the normal, horizontal placement of a definition, a vertical breakdown is often provided with components of the definition in a column to the left and explanations or comments on key words in a column to the right:

```
: DEFINITION    condition    IF this    ELSE that
    THEN continue ;
```

where:

```
: DEFINITION          Begins a new high-level dictionary entry
                      named DEFINITION .

    condition         Places a condition (non-zero/zero)
                      on the stack.

    IF                Removes and tests the condition on the
                      stack.
        this          Executes "this" if the condition was
                      true (non-zero).

    ELSE
        that          Executes "that" if the condition was
                      false (zero).

    THEN
    continue          Continues from both lines.

;                     Terminates the high-level dictionary
                      entry.
```

This expanded version is for illustration only; you will always use the horizontal format.

Additional conventions used in Forth manuals are those that Forth programmers use to make source screens readable. A full listing of Forth editing conventions for source text is provided at the end of Chapter 2. Here are three that you will observe in the examples of Chapters 1 and 2:

1. Although only one space is absolutely necessary between each word of a definition, spacing three times after a new word that is being defined sets off the major components.

2. Double spacing between phrases (logical clusters) of a definition also helps make source text legible.

3. When a definition takes up more than one line, the following lines begin with an indentation of two or more spaces to save the left margin for words being defined.

1.0 BASIC FORTH OPERATIONS

Before you begin reading this manual, please take the time to read through the Preface so that you understand the editing conventions and the relationship of the MMSFORTH USERS MANUAL to other MMSFORTH documentation.

The easiest way to learn Forth is to use it. Since Forth is an interactive language, you can and should experiment with it directly at your computer keyboard. In this introductory manual we will present many examples to illustrate the capabilities of Forth. We urge you to try these examples yourself at your computer. There are exercises at the end of most of the chapters to help you learn to use Forth on your own. Other problems may suggest themselves to you as you progress.

1.1 GETTING STARTED

Physically, your MMSFORTH system consists of a suitably configured microcomputer (typically, a Radio Shack TRS-80 Model I or III or IBM Personal Computer) with at least one disk drive and 32K RAM, and a MMSFORTH System Diskette.

Conceptually, the MMSFORTH system (see the Memory Map in Appendix A7) includes:

1. the Forth program, including interpreters, compiler, assembler, and disk (and tape) interface;

2. the basic Forth dictionary;

3. variables, buffers, and stacks;

4. and memory available for an application vocabulary to be programmed by the user.

The initial start-up procedure for MMSFORTH is outlined in detail in Appendix A1. Using these instructions, you cause the Forth System on the disk to be read into the memory of your computer. You can then use Forth immediately to solve programming problems without need for any other "monitor" or "operating system."

When your MMSFORTH System has successfully loaded, it will type out ok and space to a new line. The ok response is returned whenever the Forth text (or outer) interpreter has successfully completed your last request and is awaiting new keyboard input. You type commands at the keyboard, concluding them with a carriage return (Enter key). When the Enter key is pressed, Forth emits a space to separate your input from any generated output and then begins interpreting your commands.

Until you have actually pressed Enter, you may change your commands by pressing the Backarrow key once to delete each unwanted character (IBM: use Leftarrow key in top row), and then retyping the remainder of the line. To delete a longer series of characters, you may hold it down half a second for MMSFORTH's auto-repeat action. To delete the entire line, depress the shift key while tapping the Backarrow (i.e., "Shift-Backarrow"). MMSFORTH's optional "extended EXPECT" mode adds editor-like capabilities within the keyboard input line.

The simplest command you can give to Forth is an empty line. If you press the Enter key once, Forth will respond with a space, inspect the input, see that there is nothing to do, output an _ok, carriage return to the next line, and then wait for more input. You should try this to assure yourself that your MMSFORTH System is alive and listening to you.

1.2 WORDS

The basic command unit of Forth is called a **word.** A word consists of a string of characters (letters, numbers and/or symbols) that is delimited by spaces (or carriage returns). There are no restrictions on the characters that make up a word (except that a word may not have an embedded space, carriage return, or backspace character), and there may be as many as 31 characters in a word. This principle is important enough to summarize:

> **RULE 1: FORTH WORDS ARE COMPOSED OF UP TO 31 PRINTABLE CHARACTERS, SEPARATED BY SPACES.**
>
> Note: In general, though, we avoid using lowercase letters in Forth words. This prevents errors when the code must be run on a video display without lowercase, and it increases the readability of lowercase text and comments.

After you close a line of text with a carriage return, the Forth text interpreter scans the input line, breaking it up into words which will be executed in the order of entry. Each word in Forth has a **name** (the way you refer to it; its proper spelling) and a **definition** (the meaning; i.e., the work that is to be accomplished).

To execute a word, the interpreter must determine the word's meaning by searching the **dictionary.** The interpreter searches for the name of each word in order to locate its definition. If the word is found in the dictionary, then the definition is interpreted. If the word is not found in the dictionary, the interpreter attempts to convert the word to an integer in the current base (see next section).

When the interpreter cannot interpret a word (if the word is not found in the dictionary and is not a valid number in the current base), then an error message is given: the unknown word is echoed back to you on the video display, followed by a question mark. There is no ok or carriage return after an error message.

Words are added to the dictionary by defining a "new" word in terms of currently existing words in the dictionary. This extends the basic Forth system in the specific direction you want it to take. Rather than individual special programs, you will create a growing, more powerful Forth vocabulary. That means that you will spend more time on your first application than the second. By the time you reach your third application in MMSFORTH, you'll find that a suitable vocabulary already exists to solve most of your programming problems with very little additional effort.

1.3 **NUMBERS**

To a computer, numbers come in a variety of types: single-precision or double-precision integer, floating-point, real or imaginary, etc. MMSFORTH can handle them all, but most common work is done with integer techniques and at first we will limit our numbers to single-precision as well.

Numbers can be expressed in any base; decimal, octal and hexadecimal are standard. At any time you can use the commands DECIMAL, OCTAL, or HEX, or you can define another base to establish the appropriate way to treat all succeeding numbers, both for input and output. In general, you should pick one base and stick with it throughout all your definitions to avoid conflicts in interpretations. Initially, Forth assumes the DECIMAL mode.

Numbers may be typed in as positive (always keyed in unsigned) or negative (preceded by a minus sign) integers. **Unsigned numbers** in the range 0 through 65535 are acceptable because they can be stored in sixteen bits. **"Signed numbers"** in the range -32768 to 32767 can be accepted; the negative values are stored in two's complement form in sixteen bits. It is important to note that positive numbers larger than 32767 can be interpreted as two's complement negative numbers, especially if they are involved in arithmetic operations. Numbers larger than sixteen bits will be truncated to sixteen bits.

Numbers are entered by typing them at the keyboard. As with words, numbers are bounded by spaces. The interpreter will first search the dictionary for the Forth word entered. If it is not found in the dictionary, the interpreter tries to convert the word to a number. (Practical jokers have been known to define 1 as a word which returns a value of 5, for example!) If the number conversion succeeds, it is placed on the **parameter stack**, which will be described in the next section.

Since all numbers are stored in binary form, you can take advantage of numeric base selection to perform number conversions. To convert a decimal number to hexadecimal, for example, type:

```
DECIMAL 724 HEX .
```

and you will receive the response:

```
2D4  ok
```

Remember that you **stay** in HEX mode until you type in the command DECIMAL again.

1.4 THE PARAMETER STACK

All computer programs exist to manipulate data by using an established set of parameters. Most of the parameters that Forth words use to manipulate data are maintained on a **push-down stack**, called the parameter stack, user stack, or simply, "the stack." This stack, which is sixteen bits wide, is similar to those in pocket calculators that use postfix function keys or **Reverse Polish Notation** (RPN). A push-down stack is a particular arrangement of memory storage; Forth words that refer to the parameter stack do so by accessing only the topmost items (the ones most recently placed on the stack). Conceptually, this stack is quite like a cafeteria plate-warming stack; only the top-most "plate" is accessible, but all others remain available in the reverse of the order in which they went in. Forth utilizes your computer's random-access memory (RAM) as Last-In-First-Out (LIFO) stacks. In this text, we will often refer to the value on top of stack as TOS, the second on stack as 2OS, etc.

During the boot procedure a single pointer is initialized to point to a particular location in memory. Once this pointer is initialized, the parameter stack grows toward **low** memory. When information is to be written onto the stack, the address in the pointer is decremented and the information is then stored at the location being pointed to. When information is to be read from the stack, the information is fetched and then the pointer is incremented. Note that while reading from the stack effectively removes the information forevermore (unlike reading from conventionally organized memory), writing to the stack preserves all the prior contents of the stack that have not yet been read.

One of the basic rules of Forth, then, is:

RULE 2: MOST WORDS REQUIRE PARAMETERS ON A PUSH-DOWN STACK.

Actually, Forth manages two stacks. We will defer discussion of the other (the Return Stack) until later in the manual.

To place a number on the stack, you can type it as part of your input commands. Now type:

```
2 4 6 8
```

When you have typed in the example above, you have created a push-down stack that looks just like the entry line, with top of stack (an 8) on the right.

One of the simplest kinds of operations Forth provides for manipulating the stack is one that prints the contents of the topmost item. Forth's predefined symbol, the period or dot, causes the topmost item of the stack to be removed, converted to ASCII code digits in the current base,

and then displayed. If the stack is as we just left it, then for each period you type in, the successive topmost item of the stack will be revealed. If you type five periods followed by a carriage return, your CRT screen will look something like this:

```
. . . . . 8 6 4 2 20868 . ? Stack empty!
```

Each time the dot is encountered, the stack is depleted by one item. Since there were only four items on the stack (put there by the four operations 2 4 6 8), the fifth request for a display from the stack displays a "garbage" number from above the stack, followed by a compound error message indicating that it can't do a "." operation here, because the stack is empty.

When you receive **any** error message ("Stack empty!", "?", "Block-protect", etc.), you must remember that your stack has been emptied. Before you can perform the attempted operation, you must reenter the necessary parameters.

RULE 3: THE BREAK KEY OR ANY ERROR MESSAGE EMPTIES BOTH STACKS.

The dot operator is useful when you are debugging a definition. If you have trouble, you can display the stack, item by item, and compare the contents with what you expected. That will usually isolate the problem. Of course, reading the stack items will cause them to be removed. If you're debugging, after you're satisfied with the displayed values, you may simply type those values back in again:

```
2 4 6 8
```

The MMSFORTH System's TOOLKIT extension includes a non-destructive print-stack word, **.S** , and an even more elegant word for stack manipulation experiments, **TRY** (see the footnote at Table 3). Use them to explore the new and important world of Forth stack operations!

Access by Typing TOOLKIT
Then .S will display stack contents

1.5 ARITHMETIC

Forth has a pre-defined set of arithmetic operators (see Table 1). Since Forth uses a push-down stack and Reverse Polish Notation, parameters must be on the stack before the operation can be performed. Thus, to add two numbers together and display the results, type in:

```
5 27 + . 32  ok
```

Breaking this line down into its constituent parts, you will find that:

5	Pushes the value 5 onto the stack.
27	Pushes the value 27 onto the stack.
+	Removes the top two items from the stack, adds them together, and places the sum back onto the stack. Note that the stack has a net loss of one item.
.	Removes the top item from the stack and displays it: 32 ok

Thus you leave the stack just as it was before you started.

For programmers who have had some experience with algebraic languages (like BASIC), Forth's postfix notation may seem unusual. It will be familiar, however, to users of Hewlett-Packard pocket calculators and many office calculators, and is extremely effective when used properly. (It also happens to be the "natural" way you learned to add and multiply, before you were retrained with algebra!)

Processing of comparisons may also be unfamiliar. Forth assumes the conventions of positive logic: one (or non-zero) implies true, zero is false. The Forth relational words (such as >, <, or =) may be remembered as being written with the second stack entry on the left and the top stack entry on the right. Thus A B < will test for A < B and leave only a truth value on the stack, since both A and B have been removed. The word **NOT** (or **0=**) reverses the truth value of the top item, changing zero to one and all else to zero. Logical branching words (among others) in Forth depend heavily on these comparison methods and their resulting stack values (see Table 2).

1.6 STACK MANIPULATIONS

Other frequently performed operations are classified as stack manipulations, for which Forth provides a few simple words. These words (described in Table 3) are generally used to maintain discipline in the stack when it contains parameters. Experimenting with these words will make them useful to you quickly.

While you are practicing, keep in mind two elementary rules that you must observe with respect to stacks. The most fundamental rule is to maintain "parity" of operations on the stack: everything you put on the stack must be removed by some operation. If you leave parameters on a stack, it is because you have forgotten some step; this leads to stack overflow. In other words:

> **RULE 4: ALL PARAMETERS PUT ONTO A STACK MUST BE REMOVED WHEN THEY ARE NO LONGER NEEDED. THE ORDER WILL BE LAST IN, FIRST OUT.**

Also remember that you should never remove more items than you have first pushed onto the stack.

After you have become familiar with both the arithmetic operators and the stack manipulators, you will want to create your own combinations. For example, a convenient way to double a number is to add it to itself. This can be accomplished by the sequence:

```
DUP +
```

where the DUP is used to duplicate the number on the stack. Similarly, to square a number you use:

```
DUP *
```

1.7 DEFINITIONS

Part of Forth's power lies in your ability to define your own new words. Imagine that you frequently want to add two numbers and print the sum; you could always type in + and . for each operation. That, however, could lead to errors. Even in such a simple case it is possible to utilize Forth's power, since it would be easier to type one word instead of two.

Try defining a new word, named ADD, by entering:

```
: ADD   + . ;
```

Here is what each component in this defining operation does:

:	A colon begins a definition.
ADD	The name of the word to be added to the dictionary follows the colon that starts a definition. For reading ease the name is followed by three spaces.
+ .	The Forth words define what to do for ADD.
;	A semicolon ends a definition.

After making this definition, you merely type in two numbers and then the word ADD in order to compute and print the sum of the two numbers:

```
1 2 ADD 3  ok
4 5 ADD 9  ok
854 21 ADD 875  ok
```

Since ADD has now been defined to be identical to the executed sequence of + followed by . , you can use either ADD or the sequence to get the desired answer printed out.

What would have happened if you had used ADD before the word was defined? Forth won't allow that. It prints out the undefined word followed by a question mark. Try this with a different word, such as PLUS; you will see the error message:

```
1 2 PLUS PLUS ?
```

That points up another fundamental Forth rule:

RULE 5: ALL WORDS MUST BE DEFINED BEFORE THEY CAN BE USED.

Fortunately, especially for the novice programmer, Forth has a rich vocabulary of predefined words. One such word is **?** (pronounced "question-mark"), which prints the contents of the location pointed to by the address on TOS. ? has a simple definition which will become more clear in Section 4.2:

```
: ?   @ . ;
```

Another simple combination that is predefined in standard Forth, **2***, doubles the top stack item. If it were defined in high level, its definition might read:

```
: 2*   DUP + ;
```

Tables 1 through 3 provide easy reference to other fundamental MMSFORTH operators. The three tables include only the most basic MMSFORTH words; for a complete glossary see Appendix A9, delivered upon receipt of your MMSFORTH License Agreement.

By taking advantage of Forth's ability to define new operations, formulas may be neatly factored, with common components being defined as operator words. Making good use of predefined Forth words and choosing good names for your new operators can make the resulting definitions both compact and readable.

For instance, the stack manipulation words DROP, DUP, OVER, SWAP, and ROT can be used to assemble complex arithmetic calculations. Given the constants A, B, and C, you can define a word named QUADRATIC to compute the quadratic function $Ax^2 + Bx + C$, where x is the value on top of the stack. Once defined, it can be used to solve this quadratic equation. With values of A=2, B=-2, C=4, and x=4, and the equation factored to
(Ax + B)x + C:

```
2 CONSTANT A    -2 CONSTANT B    4 CONSTANT C
: QUADRATIC   DUP A * B + * C + ;
4 QUADRATIC . 28  ok
```

Other Forth words may be defined to plot the results for various values of x, etc.

1.8 DUPLICATE WORDNAMES, AND CATALOG

Forth will not let you use a word until it has been defined (i.e., included in Forth's dictionary), but it will accept duplicate definitions. It simply searches for the first (newest) one which fits. Because older definitions will not search the newer entries, their actions will not be affected by newer duplicate wordnames.

MMSFORTH flags each duplicate wordname as it is added to the dictionary. For example: ** Dup-name: DIR 40 1 14 ** indicates that DIR has been redefined and that it was loaded from Block 40, Line 1, Column 14. (Don't panic - MMSFORTH does this on purpose to modify a lower definition of DIR.) Duplicate wordnames can be effective; they assure that newer words will not be able to access the older definition. But a carelessly chosen one might be a poor choice for a new word. The programmer can change the new definition's wordname by re-entering the definition or the re-edited program, and also can suppress the Dup-name: display from completed application programs (see the Glossary).

Just what **is** in the dictionary? The MMSFORTH word **CATALOG** is an easy way to see. Reading from the newest entry, it displays a "Forth's-eye view" of the wordnames. See Appendix A8.1 for more information on CATALOG.

1.9 EXECUTE AND COMPILE MODES

The Forth text interpreter operates in two modes: **immediate execution** and **compilation.** In immediate execution mode each word of the input string is looked up in the dictionary and executed. During compilation, on the other hand, most words are not executed; instead a reference to them is compiled into a definition in the dictionary. The word : (colon) places the interpreter in compile mode, from which ; (semicolon) returns it to immediate execution.

The compiled form ("object code") of the definition consists of pointers to the addresses of routines that will be executed by the address (or inner) interpreter when the definition is executed. This indirect threaded form of interpretation (address interpretation) is extremely fast.

To distinguish between the modes of immediate execution and compilation, try the following examples:

905 . 905 ok	Executes immediately. Note the interaction.
: SHOW 905 . ; ok	Compiles; nothing happens yet.
SHOW 905 ok	Executes the compiled routine to produce the desired result.

There are vast numbers of variations on these basic themes, as well as whole groups of other words already defined in Forth. To learn quickly, you **must** practice with the basic Forth words, the words described in the following chapters, and the words you evolve out of experiments. You will learn to edit in descriptive wordnames, comments, stack in/out notations, initials and dates, to leave a "sketch" of what you have done.

EXERCISES
(after studying Tables 1, 2, and 3)

1. What is the difference between **DUP * DUP *** and **DUP DUP * ***

2. Using only two Forth words, define a word called **2DUP** to duplicate the top pair of stack items. That is, after:

   1 2 3 2DUP

   the stack should contain:

   1 2 3 2 3 (second 3 on top)

3. What is the difference between **OVER SWAP** and **SWAP OVER**?

Table 1 - **ARITHMETIC OPERATORS**

Word	**Description**	**Example of Stack Before** (top)	->	**Example of Stack After** (top)
+	Adds.	9 6 2	->	9 8
-	Subtracts.	9 6 2	->	9 4
*	Multiplies.	9 6 2	->	9 12
2*	Doubles an entry.	9 6 2	->	9 6 4
/	Divides.	9 6 2	->	9 3
ABS	Leaves the absolute value.	9 -6 -2	->	9 -6 2
MAX	Leaves larger of top two entries.	9 6 2	->	9 6
MIN	Leaves smaller of top two entries.	9 6 2	->	9 2
NEGATE	Performs two's complement (unary minus).	9 6 2	->	9 6 -2
MOD	Leaves modulus (division remainder).	9 11 3	->	9 2

Table 2 - **COMPARISON OPERATORS**

Word	**Description**	**Example of Stack Before** (top)	->	**Example of Stack After** (top)
<	Compares; leaves 1 if second entry less than top; otherwise 0.	9 6 2	->	9 0
>	Compares; leaves 1 if second entry greater than top; otherwise 0.	9 6 2	->	9 1
NOT or 0=	Tests for zero; leaves 1 if top entry is zero; otherwise 0.	9 6 2	->	9 6 0
0<	Tests for negative; leaves 1 if top entry is less than zero; otherwise 0.	9 6 2	->	9 6 0
=	Compares; leaves 1 if second entry is equal to top; otherwise 0.	9 6 2	->	9 0

Table 3 - **STACK MANIPULATION OPERATORS**

Word	**Description**	**Example of Stack Before** (top)	->	**Example of Stack After** (top)
.	Prints the item that is on the top of the stack.	3 2 1	->	3 2
DROP	Discards top entry.	3 2 1	->	3 2
DUP	Duplicates top entry.	3 2 1	->	3 2 1 1
?DUP	Duplicates top entry if it is non-zero.	3 2 1 or 3 2 0	-> ->	3 2 1 1 3 2 0
OVER	Copies second entry over top entry.	3 2 1	->	3 2 1 2
ROT	Rotates top three entries.	4 3 2 1	->	4 2 1 3
SWAP	Swaps top two entries.	3 2 1	->	3 1 2

General (NOT recommended for regular use!):

PICK	n PICK copies nth entry over top entry.
ROLL	n ROLL rotates top n entries.

TRY: MMSFORTH Users can load the TOOLKIT Extension, then use the word **TRY** to produce the stack operation displays on the preceding pages and many others. For example, clear the stack by pressing Break, then enter:

```
4 3 2 1 TRY ROT
```

or clear the stack and enter:

```
5 4 3 2 1   4 TRY PICK
```

(not TRY 4 PICK, because TRY must be followed immediately by the operator!).

2.0 EDITING FORTH BLOCKS

When you first brought up your MMSFORTH system, you inserted a diskette and booted up (pressed the TRS-80 Reset button, or the Alternate-Control-Delete on the IBM PC). This boot procedure read the precompiled MMSFORTH System program from the diskette into RAM. From there on, entering DIR or any of the Directory menu words will read one or more blocks of additional Forth source text from diskette and compile it into RAM. (The cassette system has to make do without the luxury of a directory-based system, but supports most operations albeit more slowly.) Compiling, in Forth, is a process which translates "user-readable" **source text** into dictionary entries that contain "computer-readable" **machine code** and addresses. Only the machine code and addresses reside in memory; source text remains on diskette.

When you simply type trial definitions at your keyboard, they are compiled immediately into your dictionary and your source text is "lost" (i.e., not preserved on diskette and not retrievable). When you reboot your system, any such occasional definition will have been cleared out. (It is this aspect of Forth that allows you to test definitions in an impromptu manner.) If you then want to use any of your occasional definitions again, you will have to type them in once more.

It is much more convenient to put your tested source text on the diskette by using an Editor, just as the MMSFORTH system programmers have done. Using the same process that boots the system (see Section 2.1.2), your definitions can be compiled into memory from the diskette rather than from the keyboard.

Before discussing how text is entered on diskette, let's consider the structure of the diskette.

2.1 BLOCKS AND SCREENS

Each diskette contains a fixed number of **blocks;** each block is numbered and contains 1024 bytes. Whenever a block is needed, it is requested by its block number. The block number reflects its relative (logical) position on the diskette(s) so that Block 0 is the first block, Block 1 the second, and Block 178 the last block on a standard TRS-80 Model III disk drive in Drive Position 0; Block 179 would be the first block on Drive 1. (Model I and IBM users read on!)

Source text in MMSFORTH is formatted for the video display in a **screen.** A screen is a set of 1024 continuous characters formatted as sixteen lines of sixty-four characters each. (That's conventional for Forth on any computer.) Since a character occupies one byte, it takes 1024 bytes or a single block to hold an entire screen. A standard Model III diskette can hold 179 screens of source text, numbered from 0 to 178. A standard Model I 35-track drive permits 87 blocks, and a 40-track single-density drive permits 99 blocks per diskette. A standard IBM PC drive (40-track, double-density, IBM format) permits 159 blocks per diskette.

Screens 0 through 14 (IBM: 0 through 19) contain your basic MMSFORTH system as precompiled binary (machine code) information. On the standard TRS-80 MMSFORTH System Diskette, Blocks 40 and 41 (53 and 54 on IBM PC) are a pseudo-directory with several menu screens appearing on higher blocks, Blocks 16-39 (21-52 on IBM PC) are Forth source for much of the code precompiled in Blocks 0-14, and a series of blocks beginning at 43 (56 on IBM) include optional Forth utilities, extensions, sample application programs, etc., all available for loading when desired. Block 15 (20 on IBM) is an Option Select Block which may be used to reprecompile a MMSFORTH System with non-standard features.

For the advanced user: Why in some cases do we use one block less than the diskette capacity? Because MMSFORTH allocates the first two to four sectors of each diskette as a **boot** section, **under** Block 0. Also, some versions of Forth use blocks the size of one disk sector (usually 128 or 256 bytes), and thus have four or eight blocks per screen. MMSFORTH avoids this confusion, and has efficient sector I/O words (**DRDSECS** and **DWTSECS**) for operations on that scale. The TRS-80 Model I and Model III use 256-byte disk sectors, and the IBM PC uses 512-byte sectors except when adjusted to M.3 format.

2.1.1 Blocks and Virtual Memory

To place a specific disk block into computer memory, Forth uses the fundamental word **BLOCK** preceded by the appropriate number. Thus,

```
35 BLOCK
```

places Block 35 in memory and leaves the memory address of the zeroth byte of the block on the stack.

If any data is written into the block in memory, the block must be marked so that the revised block contents will replace the old on the diskette -- a critical ingredient of Forth's **virtual memory** concept. The word **UPDATE** is used for this purpose. UPDATE marks the most recently used block for writing to diskette, although it may or may not be written to diskette immediately. The word **FLUSH** (or its 79-STANDARD synonym, **SAVE-BUFFERS**) is used to force any "updated" blocks out to diskette and should be used before shutting down the system.

We will often use more sophisticated words build on BLOCK, UPDATE and FLUSH, but these three words comprise the basic block I/O routines. To understand them more thoroughly it is necessary to understand the nature of virtural memory as Forth uses it. The memory address that is returned to you in the routine named BLOCK denotes the location of the data portion of one **block buffer.** The block buffers reside at fixed locations in memory, just under the MMSFORTH Dictionary and optionally at the top of memory. Each containes 1026 bytes: 1024 bytes of data from diskette plus a two-byte block ID (identification) word with a value between 0 and 65,535 (which should be enough not only for your four-drive microcomputer, but for a Winchester hard-disk drive, as well!). Your MMSFORTH System comes with two block buffers and may be configured for more. Normally, two is a good trade-off between memory usage and disk accessing.

The word BLOCK first searches the block buffer IDs to see if the block currently resides in the block buffers. If it does, no diskette access is made. Otherwise, the block is read from diskette. UPDATE sets the high-order bit of the high-order byte of the Block ID to one if the block is to be written to diskette. A block is written to diskette when its buffer is needed by another BLOCK request or when FLUSH (or SAVE-BUFFERS) is specified.

2.1.2 **Screens**

When source text is to be **interpreted** from diskette or cassette, you will use the word **LOAD.** Thus,

```
50 LOAD
```

interprets the block that makes up Screen 50 as if you had typed all the text this screen contains at the keyboard. The immediate implication is that a screen can contain both definitions **and** any immediately executable commands. Any definitions will be compiled; any other commands will be executed.

To display the contents of a screen, one can use the traditional Forth word **LIST.** For example,

```
38 LIST
```

formats Screen 38 into sixteen numbered lines of sixty-four characters each and shows them on the video display. LIST also remembers the current screen so that once you have listed a screen, you may re-List it by simply typing **L.** L looks at the contents of the user variable **SCR,** which contains the screen number of the current screen. LIST is available at **all** times on the system, unless your application removes an upper part of the system itself.

As we are about to see, MMSFORTH's **EDITOR** provides a better tool that LIST for most purposes. Using it, the equivalent commands will be 38 **EDIT,** and **E** (to re-Edit the current screen).

Here's an issue concerning LOAD which will make its point later. 79-STANDARD Forth uses **wraparound** in each block, which is to say that it treats the 16 lines as 1024 continuous characters. So don't place a character in the 64th position on one line and another in the first position on the following line, unless you intend them to be run together when the block is loaded into Forth!

2.2 THE EDITOR

********** IMPORTANT **********

The Editor is potentially dangerous in that it can **change** the code you need to have. Before you try it out, read this description. Then, experiment on a spare copy or a write-protected diskette until you have gained expertise. In particular, TRS-80 Model I users with no lowercase modification should avoid editing our provided blocks; they contain lowercase information which you cannot read. Instead, first use the ALLCAPS Utility (see Section A4.6) to convert a duplicate disk's Forth source blocks to uppercase only.

WHEN IN DOUBT, ALWAYS BACK-UP BEFORE IT'S TOO LATE!

********** IMPORTANT **********

In recent versions, the MMSFORTH Editor has evolved far beyond conventional Forth (or BASIC) editors. Some conventional Forth editing words may exist on older systems, but we will emphasize use of the simple and powerful MMSFORTH Full-screen Editor. To gain access to it, simply type:

38 **EDIT**

and MMSFORTH will invoke the Editor (which also may be done by entering **EDITOR**), will do a 38 BLOCK, will set SCR to 38, and will display Block 38's text on screen. Unlike the LIST operation, it doesn't have line numbers, so all the information fits neatly on the 1024 screen locations without scrolling off the top. Also, the cursor is changed to a Replace cursor now, signalling that you are ready to overwrite the existing text.

MMSFORTH will retain access to the Editor until you compile a definition. After compiling a definition, you need to invoke the Editor vocabulary again if you are to use it. Normally, this will only require using the EDIT operation again, or entering E if you are re-Editing the current screen. But if your application has overlayed the Editor code (by calling FORGET SCR, FORGET DIRBLK, etc.) you will have to reload it (see Section 3.2.1, "Overlays").

Within the Editor you can call on **three types of editing words:** those that work on **characters,** those that operate on **whole lines,** and those that work on **whole screens.** All require holding down an **Alternate key** or a **Control key** (the TRS-80's Clear key) while pressing the proper key. For Line and Screen operations, you must hold down the Alternate key (on TRS-80, Shift and Clear keys together). This multiple key requirement reduces accidental loss of data and permits more modes of operation, full-ASCII keyboard usage, etc.

MMS has chosen its Editor keys for mnemonic value: notice the words which accompany each key, in order to remember them easily. (MMSFORTH's NCASE structure makes it easy for advanced users to reassign these Editor key functions, too!)

2.2.1 Editing by Character

The Editor pops you onto the screen of your choice in the Replace mode, ready to overwrite one character at a time at the cursor position. On the TRS-80, the cursor (an ASCII code 143) is as high as a capital letter when in the Replace mode. (It was under the line, an ASCII 176, while in the normal keyboard entry mode.) You use the four **direction arrow** keys to steer the cursor about the screen. Holding down an arrow key will cause continuous movement after a moment of delay, and the **Shift-Rightarrow** or **Shift-Leftarrow** will hop across to the next eight-column position - try it and see. A **Shift-Uparrow** "homes" the cursor to the upper left corner of the screen. IBM users: the Replace cursor is an underline character, the **Tab** and **Shift-Tab** hop left or right by five-column positions, and a specific Home key is available.

To replace one or more characters, just type the new text over the old. On IBM use the Delete key, or on TRS-80 use **Control-D,** to Delete a character (that means to hold down the Clear key while pressing the D key, remember?); keep holding down to automatically "gobble" characters from the right. These tools are all that are required to write new screens or to rewrite old ones. And, of course, you can use the Editor to just look without changing the screen at all!

But you have more tools than these. Pressing **Control-I** (Insert on IBM) toggles the Editor from Replace to the optional Insert submode, in which you insert characters into the line instead of overwriting them. Note its tall cursor character (on TRS-80, an ASCII 191). You return from the Insert option to the default Replace submode with the same keyboard action. **Control-O** may be used to Open a place in the line by inserting one or more spaces, ASCII 32's. Unlike toggling into the Insert mode to Insert spaces, Opening may also be done in the Replace submode.

Another toggle option, **Control-P,** moves the Editor into the optional Page submode or back to the default Line submode. While in the Page submode, your editing operations operate on a continuous 1024-character "line" spanning the whole screen, instead of just the normal 64-character line. On the TRS-80, it's obvious that you are in the Page submode: its cursors are half as wide as the normal ones. The IBM Editor screen spells it out for you. When you are in the Page and Insert submodes concurrently, you have a miniature word processor which is useful for moving columns of tables, inserting code in tight areas and then realigning text, etc. (MMSFORTH also includes THE NOTEPAD, a more complete word-processor using many of these same features.)

One final character-type operation, **Control-T,** Truncates the line by replacing all characters from the cursor to the right end of the line with blanks.

Reviewing the above, note that the character editing commands require combining a control key with the action key, while cursor movement does not.

2.2.2 Editing by Line

The above operations edited one character at a time, but MMSFORTH also can move whole lines at a time. In general, full-line editing operations use the same words and concepts, except that substituting Alternate for Control (i.e., adding Shift to Control on the TRS-80) **causes the control operation to address the whole line instead of a single character.** Thus, **Alternate-D** Deletes a line instead of a character, and continuing to hold the keys down results in Deleting **all** the lines - the approved Editor method for clearing a screen - instead of deleting characters from a single line. But, and this is important, the deleted line also is stored temporarily in a scratch-pad area of RAM beginning at a variable address labelled **PAD,** and is available for reentry on the same line or anywhere else. Prove it by moving the cursor up to Line 0 and Inserting the deleted line ahead of it with an **Alternate-I.** As in the character operation, the new data is inserted **before** the present line. The IBM displays this "PAD buffer" as a separate line.

Alternate-R will Replace the present cursor line with a copy of the line currently in PAD, and **Alternate-C** will Copy the cursor line into PAD without otherwise affecting it on screen.

2.2.3 Editing by Screen

The screen is the basic unit of Forth editing, and the MMSFORTH Editor has another group of Alternates for manipulating it as a whole. **EDIT,** of course, is the word we used to Edit the screen whose block number is TOS, and we can re-Edit the current screen with E . With or without our apparent screen changes, the block will be left without actual change if we press **Alternate-Q** to Quit (**not** Break!) or if we press **Alternate-Downarrow** to advance to the next screen or press **Alternate-Uparrow** to back up onto the preceding screen - just as the downarrow and uparrow moved us to the next or preceding line, respectively.

> IBM PC: If you make the error of using Break instead of Alternate-Q to exit the editor, you will leave the display in the Editor's special window mode: 64 columns by 16 lines, centered, with scrolling disabled. To restore normalcy, just enter W/0 (window zero, the default version of SET-WINDOW) or W/0 PAGE.

To prepare the block for writing to disk we can Update the block with an **Alternate-U** before moving on. The Editor will acknowledge this operation with a brief black-out of the screen, and Forth's virtual memory will handle the rest automatically each time enough new blocks are brought in to displace the modified ones back out. Or we can use an **Alternate-S** to simultaneously Save the new screen into its block buffer (updating it in the process) and then leave the Editor. Usually, you will follow through by keying in a FLUSH command to **force** the changes to disk - a necessity when you are not modifying more blocks!

A powerful feature of the MMSFORTH Editor is its **Alternate-E** function which Exchanges the contents of the first two block buffers onto the screen. Use it to flip back and forth between any two screens of your choice while moving lines, comparing features, etc. Or continuously hold down this key combination in order to scan two supposedly similar blocks for any differences, which will flicker most obviously!

A nice Editorial touch on the TRS-80 is **Shift-Control-B**, which temporarily overwrites the upper-left corner of the block with its Block number. The IBM PC display has enough additional lines to show this Editor information normally.

2.3 INDEX

Having a superior Editor is good news, but how do you know which block is which? One way is to scan up through the screens with Alternate-Downarrow or back toward smaller block numbers with Alternate-Uparrow. But you will probably use the Forth word INDEX even more, since it provides a compact, easily scanned summary report of each block's 0-line on screen or paper. Try indexing 10 blocks starting on Block 20, as follows:

```
20 10 INDEX
```

To make optimum use of INDEX, follow the Forth convention of using the 0-line of each block for appropriate remarks.

2.4 COPY

Only one of MMSFORTH'S many copying routines, COPY, is a part of the standard MMSFORTH vocabulary. (For the others, you must load another Utility from the diskette). COPY appears to copy one block to another but actually it only renumbers the screen and prepares to send it to another place on disk, by rewriting its block ID word and marking it for Update. (That means you DON'T COPY to a number which is already in a block buffer, as that would leave two buffers with different data but identical block number.) Just enter:

```
20 40 COPY
```

to "copy" Block 20 onto Block 40. Unless you are positive that MMSFORTH's virtual memory will be invoked, follow the COPY with a FLUSH to force the actual write to diskette!

EXERCISES

1. Enter some text, such as a series of punch lines to jokes or names of friends, into each line of your practice screen.

2. Exercise until you can quickly:

   a. Exchange Lines 13 and 14 (with no duplication of text).

   b. Return your screen to its previous state.

   c. Exchange Lines 15 and 13.

   d. Exchange Lines 15 and 0. Repeat.

   e. Blank Line 15.

   f. Replace Line 5 with brand new text, two different ways.

Table 4 - **EDITING COMMANDS**

C-	means to hold the Control (TRS-80 Clear) key while pressing the active key.
A-	means to hold the Alternate (TRS-80 Shift and Control) keys while pressing the active key.
INVOKING EDITOR MODE:	
n EDIT	Enters EDITOR mode with Block n in a buffer and on the display.
E	Reenter EDITOR at current screen (SCR @ EDIT).
CHARACTER EDITING:	
arrows	Move cursor one character or one line.
Shift-arrows (TRS-80)	Move cursor right & left mod 8.
Tabs (IBM PC)	Move cursor right & left mod 5.
Home (IBM)	Home cursor to top left of screen.
Shift-Uparrow (TRS-80)	(as above)
Enter	"Carriage return" the cursor.
C-D	Delete present character, filling in from the right/down.
Delete (IBM)	(as above)
Back-arrow (IBM)	Delete preceding character, sliding text to left.
C-O	Open; insert one blank character, spreading out to the right/down.
C-I	Toggle between Replace and Insert submodes.
Insert (IBM)	(as above)
C-P	Toggle between Line and Page submodes.
C-T	Truncate present character and all to right.
WHOLE-LINE EDITING:	
A-D	Delete present line (and place in PAD); hold to clear screen.
A-I	Insert PAD's line above present line.
A-R	Replace present line with line in PAD.
A-C	Copy present line into PAD.
WHOLE-SCREEN EDITING:	
A-U	Update present screen; save display into buffer now, FLUSH to disk later via virtual memory.
A-Q	Quit EDITOR (retain last-Updated version of present screen in buffer).
A-S	Substitute present screen into RAM block buffer and leave EDITOR (Like A-U plus A-Q).
C-PgUp (IBM)	Proceed to next lower-numbered screen.
A-Uparrow (TRS-80)	(as above)
C-PgDn (IBM)	Proceed to next higher-numbered screen.
A-Downarrow (TRS-80)	(as above)
A-E	Exchange first two block buffers onto screen.
A-B (TRS-80)	Temporarily display current block# onto screen.

Table 5 - **EDITING CONVENTIONS**

Conventions exist for editing screens to make Forth source text more readable. These conventions are not dictated by the nature of Forth and will sometimes be ignored (as in tight-packed MMSFORTH System blocks). But for general use, we recommend them as good programming practice. We also recommend using MMSFORTH's NOTEPAD utility, expanded to the correct number of blocks, for **rearranging** source text within or across blocks.

1. Line 0 of each screen begins with a parenthetical comment that describes the contents of the screen. The comment identifies the screen and is conveniently listed by:
   first-block# #blocks INDEX
   You may wish to standardize with date at the left as on the MMSFORTH System Diskette, include initials, etc.

2. A single screen contains source text for words related to some one function or isolatable portion of a function. Do not put unrelated words in the same screen.

3. Do not overpack a screen. Leave several blank lines for expansion. Except on a necessarily full diskette, there is no advantage to conserving screens.

4. Do not define more than one word on a line. An exception might be two or three related constants or variables, or a couple of very brief related colon definitions.

5. Leave three spaces after the name being defined in a colon definition, to set it off from the definition.

6. Break colon definitions up into phrases, separated by double spaces, so that each phrase describes a particular operation.:
   ```
   : TRIPLE   X @  3 *  X ! ;
   ```

7. If a definition takes more than one line, indent two or more spaces on the second and succeeding lines.

8. Separate instructions in CODE definitions with two spaces.

The definitions and screens in this manual, as well as the source blocks on your MMSFORTH System, provide some good examples of well-organized Forth screens (and some tight-packed ones, too!).

3.0 COPYING, LOADING AND PRINTING

Copying operations allow us to create backup diskettes for insurance or to send appropriate programming to others. (Remember that **COPIES OF YOUR MMSFORTH SYSTEM ARE NOT TO BE SENT TO ANY OTHER USERS** - each user must buy and register his/her own licensed and serialized copy of the MMS-written portion of the system!)

Copying operations also permit the rearranging of Forth blocks on the diskette for improved logic flow, reallocation of space to new programming, and to remove those MMSFORTH System blocks from your own diskettes **before** sharing the remaining code with other MMSFORTH users.

Loading is the way we translate the information on diskette through Forth's block buffers and into executable code in RAM. MMSFORTH provides a fine selection of options for this important task.

If your computer system includes a hard-copy printer, you will like MMSFORTH's simple and versatile printer routines. They're an important professional tool for producing reports, text processing, or listings of your programs for debugging and documentation.

3.1 MORE COPYING UTILITIES

The Editor permits us to create and modify Forth source code, one block at a time. Disk operations may also require copying utilities to copy one block, multiple blocks, and multiple tracks, up to full-disk copy routines. Among the copying utilities supplied with the MMSFORTH System Diskette are **COPY, FORMAT, BACKUP** and **COPIES.** Additional routines called **DISK-TAPE** and **TAPE-DISK** are provided on the System Diskette's CASSETTE optional load blocks to permit the copying of multiple blocks from disk to cassette and vice versa.

Model III users get another major bonus: the ability to change any drive from double density format to single and back again, from the keyboard or in software. It's simple:

1 SDEN

sets Drive 1 to Single DENsity format, immediately. (And **1 DDEN** sets it back.) This means that you can assign a drive to single density and read, write or run one or more Model I Forth diskettes in conjunction with your Model III operations! Once set and loaded with the proper diskettes, the operations are completely transparent to the user.

In similar manner, IBM users may assign any Drive n to Model III format with **n M.3** and back with **n IBM.**

The MMSFORTH Editor's COPY command was described in Section 2.4. Except for COPY, all normal copying operations are accessed through the MMSFORTH System Diskette's UTILITIES menu. This is the normal default menu when you boot the MMSFORTH System Diskette. It can be recalled from other menus by entering UTILITIES or DIR.

3.1.1 FORMAT

ALL COPYING OPERATIONS IN MMSFORTH REQUIRE A PREFORMATTED DISKETTE. (By splitting the operations in this manner, we needn't waste time formatting an already-formatted diskette.) So if your destination diskette isn't already formatted, use MMSFORTH's FORMAT utility to do so before proceeding to use it. (Although MMSFORTH's FORMAT utility is not useable by other DOS systems, MMSFORTH copying operations can operate on diskettes formatted by other common DOS systems.)

To rebuild a faulty section on a diskette, first copy all retrievable blocks to another disk and then use **FORMAT-TRACKS** to save the day. To re-format 2 tracks on Drive 0 starting on Track 22:

```
0 22 2 FORMAT-TRACKS
```

3.1.2 **BACKUP**

The MMSFORTH BACKUP utility is used to make a **track-by-track copy** of one diskette to another of the same format. If you specify the same disk drive number for both copies, you will be queried for the amount of RAM available; this determines how many tracks of data can be moved across in each pass, using an internal **1DRIVE-BACKUP** routine.

When BACKUP is loaded, so is **BACKUP-TRACKS.** It's a great help when your disk drives **should** get adjusted but you've **got** to get a job done first! If your BACKUP operation fouled up a track or two, use BACKUP-TRACKS to complete the job quickly. To backup 2 tracks from Track 14 on Drive 1 to Drive 0:

```
1 0 14 2 BACKUP-TRACKS
```

WARNING: You probably got a **?Read:** message first, indicating that you have a less than perfect read in a block buffer right now. It's there so you can examine it if you wish (with EEDIT); but then **BE SURE TO CLEAR IT OUT by entering EMPTY-BUFFERS** to assure that it isn't carried on into a backup or some other "save" operation!

3.1.3 **COPIES**

The MMSFORTH COPIES utility does a slower but very versatile **Blocks-COPY.** It lets you slide a range of blocks from one place on your disk drives to another, even between two different disk formats. COPIES uses the words **BCOPY** and **<BCOPY,** which advanced users may also use directly in their own routines or from the keyboard. Following MMSFORTH's convention for calling moves of any kind, both use the stack in this order:

```
from  to  count  BCOPY
```

Thus, one would do a blocks-copy of 10 source blocks starting on Block 210 to a destination starting on Block 70 as follows:

```
210 70 10 BCOPY
```

Note that BCOPY, and therefore COPIES, automatically FLUSH to diskette. Set **PBLK,** the MMSFORTH block-protect number, low enough in advance of their use or you will have to go back and do so before proceeding!

3.1.4 **OFFSETS**

Now for a few fancy offset options in MMSFORTH.

:1, :2, :3, and **:R** are offset conversion routines which are very useful in blocks-copy operations between drives, or when you know the last block number instead of the number of blocks being copied. :1 adds to the number on TOS the number of blocks on Drive 0. In other words, it converts the block number on TOS to the correct one for Drive 1. So if you are copying Blocks 40 to 80 of one diskette to Blocks 12 and up on another, you can enter:

```
40  12 :1  80 39 -  BCOPY
```

and MMSFORTH will complete the task. Like :1, :2 and :3 convert (offset) for Drive 2 and Drive 3, respectively.

:R converts to a Range. Consider the following demonstration of its use with INDEX. As explained in Section 2.3, INDEX expects the stack to contain the first block number and the number of blocks to index. But suppose you want to index all blocks from Block 39 through Block 83, and you don't want to calculate the actual count if you can avoid doing so. :R to the rescue, as follows:

```
39 83 :R  INDEX
```

There, what could be easier? Note that :R leaves the starting block number 2OS, which can spell trouble in fancy operations (such as BCOPY) unless you think it through!

3.2 LOADING MULTIPLE BLOCKS

In Section 2.1.2, we introduced the LOAD command:

```
40 LOAD
```

loads Block 40 into your computer's executable memory, compiling the new words into the dictionary above those already aboard.

If you wish to load a range of consecutive blocks, MMSFORTH provides a number of options. First there is a word **LOADS** which expects a first block number and a number of blocks to load:

```
40 5 LOADS
```

will load five blocks, starting with Block 40. One might load a program covering Blocks 35 through 73 by entering:

```
35 73 :R LOADS
```

or load the same program from a diskette on Drive 1 with:

```
35 :1 73 :1 :R LOADS
```

= Block 35 on drive 1
To Block 73 on drive 1 — Full Range to be loaded.

But you might want to have a simple directory block with a menu, in which the operation will be identified as beginning at Block 40, without requiring the directory to know how many blocks will follow. Easy! Just prime the first block of the program with that information, by ending Block 40 with:

```
41 4 LOADS
```

NOTE! USE OF --> TAKES PRECEDENCE OVER "N LOADS"

to drag along the additional blocks!

Alternatively, you can chain successive blocks by ending each one with --> ("dash-dash-greater") to load the next. The many possible combinations of these devices make multiple block programming easier while avoiding need for a complicated file system. But be careful: --> resets the LOADS "counter", UT, to 1. Thus, if six blocks starting on Block 40 are linked together with -->'s and you use 40 4 LOADS you will load all six! But if only the first two blocks have them, 40 4 LOADS will **not** load the fourth block!

MMSFORTH handles multiple loads by moving the entire sequence of blocks through a single block buffer. This permits two-buffer cassette-based MMSFORTH systems to retain a load block in the other buffer, without requiring repositioning of the tape in order to resume operation at the load block once the other sequence is completed. Because of this feature you may wish to enter EMPTY-BUFFERS in circumstances such as swapping diskettes, to forget the earlier load block before it

could coincide with a new block you might call.

In MMSFORTH we identify the diskette's Directory Block (or the first, if there are several) with the constant, **DIRBLK.** To see the directory, just enter:

```
DIRBLK EDIT
```

This method permits **relative addressing** of programs from DIRBLK, as in:

```
DIRBLK 15 + 5 LOADS
```

If your programs are referenced in this manner, you will be able to shift the directory AND the program blocks en masse to another location using the COPIES utility, with no need to renumber block loads!

3.2.1 **And Unloading Them** (Overlays)

You cannot keep loading more Forth blocks indefinitely, or their compiled code will occupy all available RAM and overwrite the stacks, "blowing" the system. Forth traditionally defines a dummy word **: TASK ;** to mark the beginning of each temporary batch of code, or **overlay;** when you are done using that batch, it can be forgotten by saying **FORGET TASK** and all more recent programming is removed from Forth's Dictionary, freeing space for the next task!

On any system, the use of overlays insures that the definitions of each application are made in terms of the initial, standard set of MMSFORTH words rather than in terms of any new meaning that another vocabulary may have given to a particular word. This will happen automatically if you adhere to the convention of using **: TASK ;** at the beginning of each application, and **FORGET TASK** at its conclusion.

The MMSFORTH System Diskette's directory is one example of Forth's flexibility. This particular directory is done in a rather sophisticated manner which you will appreciate later, but it is just one more way to create this optional feature. The choice is yours. Often, you won't need any directory at all and will say FORGET DIR in order to free the dictionary space for other use.

3.2.2 **Look Ma, No System!**

For an unusual example of the independence of MMSFORTH and its pseudo-directory compared to a conventional disk operating system, let's boot the MMSFORTH System Diskette and then move it over to Drive 1 (leaving Drive 0 empty!) and keep using it anyway using some of that relative addressing we described.

All we need do is redefine DIRBLK to the same relative location on Drive 1, FORGET DIR to get rid of the now-wrong block numbers to load the programs listed in the menu, and call DIR again to recalculate new ones. Like so:

```
DIRBLK :1  ' DIRBLK !
FORGET DIR   DIR
```

Go ahead, try a sample program, etc. Now try explaining **that** to your non-Forth friends!

3.3 **TALKING TO YOUR PRINTER** (MEET VIRTUAL I/O!)

Before doing much work with your printer, check its features and determine whether it will run on the standard "Centronics parallel" printer port and printer-driver which are supported by MMSFORTH and your microcomputer system.

> For you technical types, MMSFORTH normally utilizes a simple Forth printer-driver (on TRS-80, with the standard DCB addresses and making slight use of the TRS-80's BASIC ROM routine). Line feed problems can usually be resolved by resetting printer switches (TRS-80 computers expect the printer to supply the LF on CR, whereas IBM PC's supply it themselves) and/or the value of MMSFORTH's LINE FEED variable (see Appendix 12).
>
> Serial and extended printer-drivers are available as options; the latter can be varied to completely bypass the TRS-80 routine, to send appropriate codes to graphics printers, etc. RS-232 serial ports and other printer-drivers are also possible to interface with MMSFORTH, although custom drivers are not delivered with the system.

MMSFORTH implements **virtual I/O**, which is to say that your MMSFORTH System doesn't much care whether its programming Input comes from tape, disk, hard disk, telephone cable, etc., or whether its Output is going to be sent to screen, printer, telephone cable, tape, disk, etc. This concept encourages a relatively independent style of programming which molds to future uses without demanding major re-programming. Normally, your MMSFORTH Disk System comes up expecting data to come and go to disk, and printing to go to screen (CRT, or Cathode Ray Tube). The I/O "switches" are reset with MMSFORTH words, and stay set one way until you again reset them or re-boot. Thus, to send the printing to your printer instead of the screen, enter **PRINT** . When you want it back on screen, enter **CRT** . And for the best of both worlds, enter **PCRT** and get both at once!

> WARNING: At the end of a PRINT operation, don't forget to enter CRT or you won't see an OK and will think you are in limbo!

3.3.1 **PLIST, PLISTS and MARGIN**

Using virtual I/O, you can power up your printer and then access it on command from the keyboard. You can list a program block by entering:

```
PCRT  40 LIST  CRT
```

Try it! Then substitute for LIST a fancier MMSFORTH option, **PLIST,** and try again. Notice that PLIST is a Printer-optimized version of LIST, beginning the listing with the block's number.

MMSFORTH offers an optional **Extended Printer-Driver** with lots of nice page-formatting features, but even its standard Printer-Driver offers one nice one called **MARGIN.** This variable contains the number of characters to indent for the left margin. (MARGIN 2+ C@ gets you the current column for the Standard or Extended Printer-Driver.) If your printer prints an 80 column linewidth and you are listing a 64-column Forth block with three extra columns for the line number, you might center the print-out by entering:

```
5 MARGIN !
```

Now print Block 47, for example, with:

```
PCRT  47 PLIST  CRT
```

Try printing an index of a range of blocks, such as:

```
PCRT  35 42 :R INDEX  CRT
```

MMSFORTH also has a multiple-blocks version of PLIST, called **PLISTS.** It's fine for listing our program blocks or your own. First properly reset your printer to the top of a page, then try PLISTS with PAGE to get back to the top of the next page when you are done:

```
PCRT  40 4 PLISTS  PAGE  CRT
```

3.3.2 **TLISTS, TINDEX and SCREEN-PRINT**

MMSFORTH also provides time and date capabilities in its **CLOCK** extension. Once this is loaded and initialized with **SET-TIME** and **SET-DATE** , a still fancier blocks-printing routine called **TLISTS** becomes available. It is a truly professional way to document your programming, complete with date and time on a page-numbered title-line. An example follows:

```
12 25 81 SET-DATE  14 32 45 SET-TIME   5 MARGIN !
PCRT  20 45 :R  TLISTS  PAGE  CRT
```

TINDEX is a similar utility for combining the same title-line with INDEX. TINDEX and TLISTS reset the page number to 1 as they begin their count. If you prefer to continue numbering where you left off, use +TINDEX and +TLISTS, instead. You can start these at a page number of your choice, by storing the desired number into TPAGE before calling them. Unless you are sure you have completed a page, you should call 0TPAGE first, to re-zero the program's line counter:

```
0TPAGE   12 TPAGE !   PCRT  30 10 +TLISTS  PAGE  CRT
```

will print a titled Page 12 and consecutively number following pages as needed, for blocks starting with Block 30. When it is done, it will considerately PAGE the printer to the top of the next sheet.

MMS thoughtfully provides a title line in the CLOCK utility; you can use it to print out an elegant listing of our system. Initialize the CLOCK settings as outlined above, then type in this line:

```
PCRT  15 72 :R TINDEX  PAGE  73 131 :R +TINDEX  PAGE  CRT
```

This will produce a neat, titled Index of the TRS-80 Model III block headings for your reference. Note that empty, formatted blocks and blocks filled with machine code will be listed as empty.

Would you like a complete listing of the MMSFORTH source code on your diskette? If so, insert a **lot** of paper and plan to go off for lunch! Start with just a few blocks anyway, for practice. Now, consulting the new Index we just printed above, specify an appropriate range of block numbers which may differ from this example:

```
PCRT  15 117 :R TLISTS  PAGE  CRT (for TRS-80 Model III)
PCRT  20 136 :R TLISTS  PAGE  CRT (for IBM PC)
```

We had to start the second page of the Index with +TINDEX, but notice that TLISTS, which knows you are sending it regular blocks of 16 lines each, inserts carriage returns in the right places to do this itself.

You may prefer not to list your own source blocks as copyrighted by MMS (and MMS may prefer it, too!). For regular use, just change the final line on the CLOCK blocks so your text follows TITLE. But if you are just dashing it off, you can reset TITLE as you go:

TITLE This is my OWN source code! (Enter)

That's all, it's ready to go for your TLISTs or TINDEXes or whatever! Note that TITLE's maximum phrase length, less the date and time information, is limited to 42 characters unless you change the source code.

A final trick for your printer: load MMSFORTH's **SCREEN-PRINT** extension for some screen-printing fun! Now you can press **Alternate-*** on the TRS-80 or **shift-PrtScr** on the IBM to print a "snap-shot" of the current screen. Try this for documenting some of your initial Forth experiments for later analysis.

If your printer can print the computer's graphics characters, see the SCREEN-PRINT source block for the appropriate modification. Otherwise, graphics characters will be printed as dots by SCREEN-PRINT, and as blanks by the other MMSFORTH routines.

If you are using a TRS-80 Model I without hardware lowercase modification, you're out of luck; SCREEN-PRINT won't work.

EXERCISE

Blank Lines 1 through 15 of your practice screen.

Add the conventional phrase to the first line that will make this screen a sample overlay.

Using such Forth words as the arithmetic operators (Table 1), the stack manipulators (Table 2), and others from Appendix A, create at least three new words.

Enter your definitons of 2DUP and the new words into the practice screen.

Review Table 5, "Editing Conventions," and check the screen.

Load it.

Test and debug it.

4.0 DATA DECLARATIONS

Frequently it is useful to set aside cells in memory to reserve constants, variables, and arrays. There are a series of words in Forth that allow you to allocate these types of data structures. This chapter discusses the appropriate commands.

4.1 SIXTEEN-BIT (SINGLE-PRECISION) CONSTANTS

If a value is used frequently or if a value is associated with a specific function, you might want to name it. Often the name is easier to recall and enter correctly than is the number itself. A named value is a constant and **CONSTANT** is the Forth word used to assign dictionary names to constants. For example, if you are converting miles to feet, you can define a CONSTANT named FT/MILE. Thus,

```
5280 CONSTANT FT/MILE
```

creates the new word FT/MILE and assigns it the initial value 5280, which CONSTANT found on the top of stack.

After FT/MILE has been defined, you can use it just as you would 5280 to place a value on the stack. That is, if you type FT/MILE, the value 5280 will be placed on the stack. The phrase:

```
FT/MILE 3 *
```

computes the number of feet in three miles. Once a value is defined as a CONSTANT, its binary value is independent of the current number base.

4.2 SIXTEEN-BIT (SINGLE-PRECISION) VARIABLES

A value which changes frequently is called a variable. The Forth word **VARIABLE** names a location whose value is likely to change. For instance, you might want a variable to keep track of the number of customers who have walked into a new store. You can do that with a statement like:

```
VARIABLE PATRONS
```

which means "define a variable named PATRONS, automatically initialized with a value of zero". (Other versions of 79-STANDARD may **not** initialize it to 0!)

When you invoke a constant by its name, its **value** is placed on the stack. Invoking a variable, on the other hand, places its **address** on the stack. After placing the address of PATRONS on the stack, you will sometimes wish to obtain the contents of PATRONS. The Forth word **@** (called "fetch") replaces the address on the stack by the contents of the two bytes at that address. To get the current number of customers into the top of the stack for processing, you write:

```
PATRONS @
```

Sometimes you need to examine the contents of a variable. The Forth word **?** (question-mark) puts the current value on the stack and shows it:

```
PATRONS ? 0  ok
```

The word **!** (called "store") is used to store a sixteen-bit value into a location. ! uses the value, which is the second item on the stack, and an addresss to store into, which is on top of the stack, to alter the contents of a VARIABLE. If you write,

```
5 PATRONS !
```

Forth will store the value 5 into the VARIABLE named PATRONS. Prove it:

```
PATRONS ? 5  ok
```

To put whatever value is currently on TOS into a specified location, then, you merely need to specify the address (by name) and invoke the ! operator:

```
(variable-name) !
```

The Forth word **'** (pronounced "tick", interpreted as "address of...") can be used to change the value of a CONSTANT:

```
5000 ' FT/MILE !      FT/MILE . 5000  ok
```

From this, you can see that CONSTANT and VARIABLE may be used for the same jobs; but the former is more **efficient** for putting its value on stack, the latter for modifying it.

The Forth word **+!** (called "plus-store") **adds** a new value to a variable:

```
101 PATRONS +!        PATRONS ? 106  ok
```

In the same manner as !, +! increments the contents of the item whose address is on TOS by the second item on the stack.

To copy data from one place in memory to another, say from the variable OLDPATRONS to PATRONS, you can use the sequence:

```
OLDPATRONS @  PATRONS !
```

Set up your own variables and use these operators (@ ? ' ! and +!) until you understand how each works.

One of the surprising aspects of Forth programming is how few constants or variables are needed. Since the parameter stack is used to hold values which need not be named or need not take up dictionary space, the need to define every literal or temporary value as a constant or variable is eliminated. Only fundamental parameters in an application will need dictionary space.

The most common failing of inexperienced Forth programmers is excessive use of constants and variables. To realize the most value from your MMSFORTH system, try to be alert to this tendency and resist it. But also resist piling too many values on stack at once, which makes managing and visualizing the stack unnecessarily difficult.

4.3 BYTE (CHARACTER) VARIABLES

Just as @ and ! transfer data in sixteen-bit units between the stack and memory, the Forth words **C@** ("character-fetch") and **C!** ("character-store") transfer data in eight-bit, single- character bytes. Eight-bit numbers occupy the low-order half of a stack entry. C@ fetches a single byte from the location specified by the TOS and puts a zero into the high-order byte (to fill out the stack entry). C! takes the low-order byte of the second stack item and stores it into the byte addressed by the top item, deleting both items from the stack. **C?** ("character-questionmark") is the one-byte equivalent of ?.

As one might expect on eight-bit microprocessor systems, the character fetch and store operations are both faster and more conservative of memory than their sixteen-bit counterparts. This makes a noteworthy difference between Forth and other high-level languages. Forth does not discriminate between data types by context but rather by the operators that are used to manipulate the data. Thus a sixteen-bit named variable could contain either two characters of a word, or two eight-bit binary numbers (such as a byte vector), or a sixteen-bit binary number. Its usage depends upon the operators that you choose to manipulate its data. This method produces more readable definitions, more efficient execution, and more flexible programming.

Single-character constants and variables can be declared in a manner similar to their sixteen-bit counterparts by using the Forth words, **CCONSTANT** and **CVARIABLE.** Just as CONSTANT does, CCONSTANT needs an initial value on the stack, followed by the name being defined. And CVARIABLE, like VARIABLE, automatically initializes with a value of zero. The space used, however, is only one byte wide, which limits you to numbers in the range 0 to 255. Often this range is more than you need. If you wanted to keep track of the current channel number to which a particular TV set was tuned, you could use:

```
CVARIABLE CHANNEL  n CHANNEL C!
```

where n represents the initial channel number. (The number of TV channels would never exceed 255.)

After a CVARIABLE is defined the operators C@ and C! may be used on them. Mixing C@, C!, @, ! and +! between definitions is perfectly legal. Be sure, however, that you understand exactly what result you intend to achieve.

4.4 ARRAYS

Arrays of data items are important in many applications. For example, instead of handling a set of ten different temperature readings as T0, T1, ..., T9, it would be better to use ten successive data elements named TEMP. Through suitable addressing arithmetic Forth can compute the requisite element's address. This is more flexible to program as well as more economical of dictionary space.

MMSFORTH provides an unusually complete ARRAYS extension which can be loaded into the dictionary as a set, or from which appropriate lines may be incorporated. It and other optional extensions include one- and two-dimensional byte-, single-, and double-precision array capabilities, addressing integer and floating-point numbers as well as alphanumeric strings.

To utilize MMSFORTH's single-precision integer array capability, load the ARRAYS extension and enter:

```
9 ARRAY TEMP
```

Now TEMP has 10 elements, 0 through 9 (**not** necessarily initialized to contents of 0). To reset the contents of the fourth element to 2000:

```
2000 3 TEMP !
```

To set Element 8 to twice Element 4:

```
3 TEMP @  2*  7 TEMP !
```

And to display its new value:

```
7 TEMP ?
```

Instead of loading the entire ARRAYS source code block, one might choose to extract just the necessary lines from it. But beginning Forth programmers are warned that some of the colon word definitions in this block require the preceding assembler code.

4.5 OTHER MEMORY OPERATIONS

The words **ERASE, CMOVE, <CMOVE, MOVE, BLANK,** and **FILL** can be used to manipulate memory locations.

ERASE is used to zero a region of memory:

address length ERASE

zeroes the region that begins at the address specified, for the specified length given in **bytes.** (Some TRS-80 Model I lowercase modifications shift the displayed values of ASCII Codes 0-31 decimal by 64, causing the video display to show these 0 characters as @ characters! To the computer, they are still zeroes.)

BLANK works similarly, setting the memory area specified to blanks (ASCII 32).

CMOVE is used to transfer a region of memory to another location:

source-address destination-address #bytes(count) CMOVE

moves the number of bytes specified, beginning at the source address, to the destination address. The contents of the destination region are overwritten; the source region remains the same.

<CMOVE (pronounced "reverse-cee-move"), works equivalently except that it moves the bytes starting at the high end of the range - a necessary maneuver if the source and destination ranges overlap!

Because they are destructive overwrites into memory (where the dictionary resides), these words must be used extremely carefully. When a region of memory is specifically reserved, however, as with arrays or block buffers, ERASE and CMOVE can be used to initialize arrays or to copy arrays from one place to another.

In the example used above,

0 TEMP 20 ERASE

clears (zeroes) the temperature array.

Defining a second array:

```
9 ARRAY 2TEMP
```

and using CMOVE :

```
0 TEMP   0 2TEMP   20 CMOVE
```

would copy the array in TEMP into 2TEMP.

MOVE is similar to CMOVE, but moves in 2-byte (16-bit word) chunks, and its count is a **word** count.

FILL is the most general way (BLANK and ERASE both use it) of setting an area to a specified value:

```
starting-addr  #bytes  fill-char  FILL
```

EXERCISES

1. Define EXCHANGE to exchange the contents of two variables. That is, if A and B are variables, then the result of the command A B EXCHANGE should be to place the value of A in B and the value of B in A.

2. Define TRANSFER to move data between two arrays of the same length. (Define a CONSTANT to specify a length.)

3. Using the arrays defined above, clear the first array and TRANSFER the initialized array to the second array.

Table 6 - **MEMORY OPERATORS**

Word	**Description**	**Example of Stack Before** (top)	->	**Example of Stack After** (top)
@	Fetches the contents of the item whose address is on the TOS.	20000	->	257
!	Stores the 2OS item into the location whose address is TOS.	3 20000	->	
?	Fetches, prints the contents of the location whose address is TOS.	20000	->	
+!	Increments the location whose address is TOS by the 2OS item.	101 20000	->	
'	Places the address of the following word onto the stack (Use: ' word).		->	26712
C@	Fetches a **byte** whose address is TOS. The byte is right-justified on the stack.	20000	->	1
C!	Stores a byte into location whose address is TOS. Only right-most byte is stored.	254 20000	->	
C?	Fetches, prints the byte in location whose address is TOS.	20000	->	
DP	Points to next available byte in the dictionary. (DP is a variable.)		->	27014
HERE	Places address of next available byte in dictionary on stack.		->	35024

Table 6 - **MEMORY OPERATORS** (cont.)

Word	**Description**	**Example of Stack Before** -> top	**Example of Stack After** top
ERASE	Zeroes memory at address 2OS for #bytes specified. (Use: addr count ERASE)	21200 25 ->	
BLANK	As above, but blanks.		
CMOVE & <CMOVE	Moves bytes from one location to another: (source-addr dest.-addr count CMOVE)	20000 22514 256 ->	
MOVE	As CMOVE, but 2-byte steps.		
FILL	General form	15360 1024 65 ->	

IBM PC: Equivalent long-address words, for RAM addresses greater than 65,535, are provided in the LONG-ADR Extension (see Appendix A5.10).

5.0 HANDLING TEXT

MMSFORTH includes various ways to handle the portrayal of text, including a STRINGS extension and some built-in words for less sophisticated operations on words and numbers.

5.1 CHARACTER STRINGS ("dot-quote")

Forth uses the word **."** ("dot-quote") to open a character string of text, which is closed with the next " or at the completion of the screen. Note that ." must be surrounded by spaces, like any other Forth word; however, its closing quote is in-line unless a closing space is desired in the quote itself.

Unlike some versions of Forth, MMSFORTH has an **intelligent** dot-quote. That means it is able to determine whether it is in a compiling definition or the immediate execution mode, and it will adjust for proper operation in either case. Very intelligent, indeed!

Let's use ." in a simple example:

```
: HELLO    ." HI THERE, PROGRAMMER!" ;
```

Once you have entered (compiled) this new word, test it by entering HELLO. It works, but notice that the following _ok isn't the most professional way to set off the line! We can improve its readability with a terminating **CR** to force a Carriage Return to a new line for the offending _ok:

```
FORGET HELLO    : HELLO    ." HI THERE, PROGRAMMER!"  CR ;
```

Try this newer definition now. Better?

Let's continue to refine this operation by adding a **PAGE** command to clear the display, then using **PTC** to PuT the Cursor a distance down and to the right:

```
FORGET HELLO
: HELLO    PAGE  8 20 PTC  ." HI THERE, PROGRAMMER!"  CR ;
```

Try it, you'll like it! To see the classiest effect yet, use your computer display's lowercase characters for the quote. A Shift-zero on TRS-80 or CapsLock key on IBM PC toggles you into or out of the lowercase mode.

5.2 THE STRINGS EXTENSION

MMSFORTH's STRINGS extension contains a powerful set of string operators which are quite similar to those in Radio Shack/Microsoft Extended BASIC. But before using them, let's consider how MMSFORTH views a string in RAM:

> **A STRING IS UP TO 256 CONSECUTIVE BYTES OF COMPUTER MEMORY WHICH ARE A COUNT BYTE FOLLOWED BY THAT MANY BYTES, EACH CONTAINING THE APPROPRIATE ASCII CODE FOR A CHARACTER IN THE STRING.**

Armed with this valuable information, load the STRINGS words from your MMSFORTH System Diskette by entering:

```
EXTENSIONS  STRINGS
```

Now let's create a new **string-variable** called HELLO:

```
30 $VARIABLE HELLO
```

Here we allocated a 30-byte maximum size for the string to be called HELLO. If you don't create the variable, you won't have anywhere to store the string, except in PAD on a temporary basis. Now you might use IN$ to input the string to PAD and thence to HELLO:

```
IN$  HELLO $!
```

This is quite a bit like the storing of single-precision numeric variables, in Chapter 4.

Or, you might use a **string-literal** to assign the string inside your program, like this:

```
$" Hi there, programmer!"  HELLO $!
```

Always be sure to store the string literal **before** any other operation overwrites it in the unprotected "scratchpad" area of computer memory (called PAD).

Now, we have a string "HI THERE, PROGRAMMER!" consisting of a series of bytes in memory beginning at an address which is put on stack when we say HELLO. The first byte of this string contains its length, 22 bytes. You can prove this by entering:

```
HELLO  C? 22  ok
```

or using a fancy STRINGS word, **LEN**, which does exactly the same thing itself (a C@) plus a . :

```
HELLO  LEN . 22  ok
```

Printing a string works about like a numeric constant, also. Instead of the word "dot" it uses the word "string-dot":

```
HELLO  $. Hi there, programmer! ok
```

The following Forth program is shown in the LIST format, with line numbers on the left (don't enter them!). It exercises some MMSFORTH STRINGS words to display any short word in a square format of repeating lines, with a separator character slicing diagonally through the display. Try it, modify it, and think about it!

Listing 1 - **STRINGS EXAMPLE**

```
 0 ( 06/01/81 STRINGS example for MMSFORTH MANUAL )  : TASK ;
 1 50 3 LOADS ( load STRINGS; may start other than Block 50.)
 2 12 CONSTANT MAX-LENGTH   MAX-LENGTH $VARIABLE NAME
 3 : SIZE-OK?   PAD DUP COUNT SWAP DROP  MAX-LENGTH >
 4   IF  ."  - Pick a shorter word!"  DROP 0
 5   ELSE 1
 6   THEN ;
 7 : PRINT-IT  BEGIN  CR ." Enter what word"  IN$  SIZE-OK?
 8             UNTIL   NAME $!  NAME LEN 1+ 0
 9   DO  CR  NAME I LEFT$ $.  ." *"
10      NAME DUP  LEN I -  RIGHT$ $.
11   LOOP  CR CR ;
12 : STRING-DEMO   BEGIN  PRINT-IT  ." Again" Y/N  UNTIL
13   ."    - OK, see you soon!"   10000 0 DO LOOP   CR ;
14 STRING-DEMO   FORGET TASK   DIR
15
```

Consult your MMSFORTH Glossary (supplied in the additional Appendix materials sent upon return of your User License Agreement) for a complete set of STRINGS words and further examples of their use.

5.3 "PRETTY" NUMBERS (.R)

Using . and ? , we have been printing numbers as they come. But it often is desireable or necessary to line up our numbers in neat columns or otherwise organize their format. The easiest Forth trick for this is **.R** ("dot-R"), which prints the number in a field of the width specified on TOS. To compare it in action, try each of the following:

```
: RANGE       CR  1+ SWAP  DO  I .     LOOP ;
: NEAT-RANGE  CR  1+ SWAP  DO  I 4 .R  LOOP ;
```

Now test each by saying 1 25 RANGE and 1 25 NEAT-RANGE , etc. See how .R keeps things neat if you plan your column sizes appropriately? We'll look into that "DO ... LOOP" structure later in Section 6.5.

5.4 MIXING NUMBERS AND STRINGS

Of course one can combine numbers and strings to make compound words or phrases such as 2-DRIVE SYSTEM, LINE #4:, etc. But to do so effectively, you must understand that in Forth (MMSFORTH and other 79-STANDARD versions), numbers are printed somewhat differently than in BASIC: **NUMBERS NORMALLY PRINT WITH A TRAILING SPACE BUT WITHOUT A LEADING SPACE.**

Thus, with the number 4 on TOS, we might print "Line #4" like this:

```
: LINE#   ." Line #" . ;
4 LINE# Line #4  ok
```

However, adding a closing colon to this print-out to show "Line #4:" introduces a problem:

```
: LINE#   ." Line #" .  ." :" ;
4 LINE# Line #4 : ok
```

which has an unwanted space between the last two characters. The magic phrase for fixing it is **0 .R** :

```
: LINE#   ." Line #" 0 .R  ." :" ;

4 LINE# Line #4: ok
```

How about that!

5.5 **PICTURED NUMBER FORMATS** (<# # # #>)

This is just a teaser for now. But later in our CHECKBOOK demo program we will be using special number formats of our own design, to display the values neatly with dollar signs, two digits to the right of a decimal point, etc. Forth can create as many special pictured number formats as you like, but you must remember that they expect **double-precision** inputs from the stack.

6.0 CONDITIONAL BRANCHES AND LOOPS

Many definitions execute words (made up of other definitions), one right after another. However, it is often necessary to alter the way in which words are executed in a definition so that the routine may respond differently to different input conditions without being re-typed. This is accomplished by using the Forth structures for loops and conditionals. Loops cause a sequence of words to be repeated a specified number of times; conditional structures allow the application to choose a sequence of words based upon a given test (or condition). The following words **compile** logic which alters the execution of words within a definition:

IF ... ELSE ... THEN

DO ... LOOP or DO ... +LOOP

BEGIN ... UNTIL

BEGIN ... WHILE ... REPEAT

ACASE ...." ... ... OTHERWISE ..... CASEND

NCASE . . . " ... ... OTHERWISE ..... CASEND

IMPORTANT: NO loop or conditional structure can be executed directly from the keyboard without being included inside a definition. The control words listed above are designed to compile appropriate logic control and thus are **meaningless** if used outside a definition.

Remember this rule:

> **RULE 6: COMPILING WORDS MUST NEVER BE USED OUTSIDE A DEFINITION.**

This chapter contains discussions of various conditional structures, loops, and related matters. Each discussion is capped by appropriate exercises.

6.1 CONDITIONAL BRANCHES

Three compiling words, **IF**, **ELSE**, and **THEN** are used to compile conditional branches in a definition. In Forth, conditional branches examine the top of the stack to decide which branch wil be taken. A conditional branch has the following structure:

```
: DEFINITION    condition    IF this  ELSE that  THEN continue ;
```

where:

```
: DEFINITION
    condition       Places a condition (non-zero/zero) on stack.
    IF              Removes and tests the number on stack.
        this        Executes "this" if the number was
                    non-zero (true).
    ELSE
        that        Executes "that" if the number was zero
                    (false).
    THEN
    continue ;      Continues from both lines.
```

IF marks the place where the top of the stack is popped and examined; if the value is non-zero, everything up to ELSE is executed - at ELSE, execution skips to THEN. On the other hand, if the stack value is zero everything up to ELSE is bypassed and everything after ELSE up to THEN is executed.

RULE 7: EVERY IF MUST BE FOLLOWED BY A THEN.

For example, you could print non-zero numbers if you defined NON-ZERO in this manner:

```
: NON-ZERO
    DUP             Duplicates the number.
    IF              Tests and discards the top number.
        .           Prints it if it is non-zero.
    ELSE DROP       Otherwise drops it.
    THEN ;
```

It was necessary to DUP the number on the stack prior to the test because IF removes the number it tests.

The ELSE clause is optional. For example, to increment the TOS only if it is non-zero, you can define INC:

```
: INC
    DUP             Duplicates the number to save it.
    IF              Tests it.
        1+          Increments it if it is non-zero.
    THEN ;
```

The truth value on the stack is often the result of a comparison that uses one of the Forth words, <, >, or =. These operators test the top two stack items for the relations "less than," "greater than," or "equal to," respectively. <= ("less than or equal to"), >= ("greater than or equal to"), and <> ("not equal to") are also supported in MMSFORTH. Each removes the top two items which it tests, leaving one for true and zero for false. (Review Table 2.)

< Leaves one if the 2OS item is **less than** TOS.

= Leaves one if the 2OS item **equals** the TOS.

> Leaves one if the 2OS item is **greater than** TOS.

The comparison operators observe the same Forth convention of Reverse Polish Notation that arithmetic operators do. All Forth operators retain their conventional meaning. Thus,

```
9 5 <   is the same as    9 < 5     i.e., false.
4 2 >   is the same as    4 > 2     i.e., true.
```

For example, suppose an input data item (placed on stack by INPUT) is to be regarded as either a decimal digit (if it is < 10) or as a code for an analog function (if it is >= 10). Then the definition:

```
: DECIDE   INPUT  DUP 9 >  IF DO-FUNCTION
      ELSE DIGIT !  THEN ;
```

either does the analog function if the INPUT is greater than nine or else saves the value in the variable DIGIT.

Two additional comparison operators are **0<** and **0=**, which may be defined:

```
: 0<   0 < ;
: 0=   0 = ;
```

to test for negative or equality to zero, respectively. (Actually 0< and 0= are defined in machine code while <, >, and = are defined in terms of them.) These operators also replace a single argument by a truth value.

To negate a condition, use the Forth word **NOT**, which replaces zero by one, and replaces any non-zero value by zero. Because this is identical to the action of 0=, the definition of NOT is just:

```
: NOT   0= ;
```

In the following example we include a specific test for zero before storing a data item:

```
: ITEMS   DUP  IF DATA !  ELSE DROP  THEN ;
```

where:

```
: ITEMS              (The upper limit is on the TOS at entry.)
   DUP IF            Tests for the non-zero upper limit.
       DATA !        Stores the value into the variable DATA
                     if it is non-zero.
   ELSE DROP         Discards any unwanted zero.
   THEN ;
```

Since it is common to ignore zero values during an operation, Forth provides the word **?DUP.** It duplicates the TOS only if it is non-zero. Using **?DUP**, the definition of ITEMS becomes shorter:

```
: ITEMS   ?DUP  IF DATA !  THEN ;
```

In other words, ?DUP eliminates the use of the phrase ELSE DROP. The definition of ?DUP, by the way, is just:

```
: ?DUP   DUP  IF DUP  THEN ;
```

6.2 COMBINING TRUTH CONDITIONS

Sometimes it is useful to combine several truth values. For instance, you may want to execute statement X only if both parameters on the stack are non-zero. Although this is a trivial example, it serves to demonstrate that two conditions can be met in one definition. The logical operators found in Table 2 are used in combining truth definitions.

The word **AND** performs a logical AND of the top two stack items (bit by bit). This can be used to define compound conditions. For example, if FROM and TO are constants, then you can define BETWEEN:

```
: BETWEEN
    FROM OVER <   Compares the number with FROM .
    SWAP          Swaps the truth value with the number
                  to be tested.
    TO <          Compares the number with TO .
    AND ;         Takes the AND of the two truth values.
```

BETWEEN determines if the top stack item is between FROM and TO, exclusive.

MMSFORTH also supplies an **OR** word, even though the logical OR function can usually be handled by addition.

Truth values are really no different from numbers and may be used arithmetically. Consider this example, which computes (the characteristic of) the base-ten logarithm of a number:

```
: LOG
    DUP
    10 >=          Leaves one if n >= 10.
    OVER 100 >=    Leaves one if n >= 100.
    +              Adds the running sum of the truth values.
    OVER 1000 >=   Leaves one if n >= 1000.
    +              Adds to the running sum.
    SWAP 10000 >=  Leaves one if n >= 10000.
    + ;            Adds to the running sum.
```

EXERCISE

1. Given the constants FROM and TO, define a word named OUTSIDE that will leave true on the stack if the top item does not fall between FROM and TO.

6.3 INDEFINITE LOOPS

All loops are governed by the values on the stack. Here is the structure for an indefinite loop (words in lower case represent your application's named and tested definitions):

```
: EXAMPLE   BEGIN  process  condition  UNTIL  continue ;
```

where:

: EXAMPLE	Creates a dictionary entry for the new word, EXAMPLE.
BEGIN	Marks the beginning of an indefinite loop.
process	Defines the action(s) to be executed one or more times.
condition	Leaves a truth value on the stack, either zero for false or non-zero for true.
UNTIL	Pops the value off the stack, returning to BEGIN if the condition is zero.
continue ;	Continues execution after the loop ends.

BEGIN marks the beginning of the loop. The **body** of the loop (here indicated by the words "process" and "condition") is executed each time through the loop. The body of the loop must leave a numeric value on top of the stack; that value is examined each time the UNTIL statement is reached. If the value on top of the stack is zero (false), the loop is repeated; to terminate the loop, any non-zero value (true) is placed on the stack. Thus, loop repetition is directly under program control. When the loop is ended, the word following UNTIL will be executed. UNTIL removes the number it tests from the stack.

Suppose #ARRAY is the starting address of an array of sixteen-bit entries containing at least one non-zero entry. You can find the address of the first non-zero entry by the loop:

```
: FIND#0   #ARRAY 2-  BEGIN 2+  DUP @  UNTIL ;
```

where:

: FIND#0	
#ARRAY 2-	Decrements the array address by two.
BEGIN	Begins an indefinite loop.
2+	Increments the address by two.
DUP @	Fetches the contents while saving the address.
UNTIL ;	Ends the loop at the first non-zero entry.

In the body of the loop, 2+ increments the address on the stack before the examination of the contents of the address. Consequently, you must decrement the address #ARRAY with the phrase 2- before entering the loop. The loop terminates when a non-zero entry is found. Notice that DUP preserves the address on the stack during the loop and that, once the loop is complete, the last address remains on the stack.

It is possible to create a simple program that will execute forever:

```
: FOREVER   BEGIN  whatever  0 UNTIL ;
```

The zero preceding UNTIL guarantees re-execution of the loop body.

Often you will wish to execute some phrase a specified number of times (say, ten). That can be done by writing this BEGIN ... UNTIL sequence:

```
: TEN-TIMES  0 BEGIN  phrase 1+  DUP 10 =  UNTIL DROP continue
;
```

where:

```
: TEN-TIMES
     0              Places a counter on the stack.
     BEGIN
       phrase       Provides the action(s) to be repeated.
       1+           Increments the counter.
       DUP 10 =     Tests for equality to ten.
     UNTIL
     DROP           Discards the counter.
     continue ;
```

Initially, the top of the stack is zero; after the body of the loop (the useful work) is performed, the counter at the top of the stack is incremented and compared to ten. If ten has been reached, the UNTIL word will cause control to "drop through" to the next word **without** performing again (unlike BASIC). Otherwise, control will return to the BEGIN at the start of the loop.

Notice that there are seven words that must be executed (five of them repeatedly) to implement this loop. Also, if the body of the loop needs to add or remove successive items from the stack, the counter at the top of the stack must be accounted for and operated around. In this case a controlled loop would serve your purpose better. Controlled loops are discussed in Section 6.5, after consideration of the return stack, which is frequently needed for the initialization of controlled loops as well as other operations.

6.4 THE RETURN STACK

As explained in Chapter 1, all Forths use at least two stacks. The most visible stack is the **parameter stack**, which is used to manipulate parameter values and memory locations. The second stack, the **return stack**, is used primarily for program control. Values saved on this stack include return addresses for colon definitions and counters for controlled loops. The two stacks segregate parameters from program control values so that Forth code is both more readable and debugged easily.

Forth's use of the two-stack architecture came about because of the hazards inherent in conventional single-stack programs in which parameters, program addresses, and other control information are all combined together in the same stack. In such a system, operations that should only be concerned with parameters must keep track of other entries in the stack. When two stacks are used, parameters and return addresses need never be confused.

There are occasions, however, when something on one of the stacks would be useful if available on the other or when one component of a definition requires one or more numbers that would otherwise be buried in the parameter stack. The basic FORTH vocabulary therefore includes three important words for transferring data from one stack to another:

>R Pops a number off the parameter stack and pushes it **onto** the return stack.

R> Pops a number **off** the return stack and pushes it onto the parameter stack.

R@ Copies the number that is on the top of the return stack and pushes it onto the parameter stack, without changing the return stack. (See Section 6.5 for an example of usage).

>R and R> enable you to use the return stack as an auxiliary stack. For example, you may transfer a number to the return stack prior to a calculation which makes heavy use of the parameter stack. Since the number is on the return stack, you can fetch it back without disturbing the parameter stack. Judicious use of >R and R> can make definitions more readable.

Because the return stack is primarily used to hold control values, there are two important constraints on your use of it, the first of which is given here. Since the second constraint concerns DO ... LOOPs, it is given as Rule 9 (in Section 6.6).

RULE 8: ANYTHING PUSHED ONTO THE RETURN STACK MUST BE REMOVED WITHIN THE SAME DEFINITION.

For example, if you define CRASH this way:

```
: CRASH   0 >R ;
```

and then try executing CRASH, you will crash because the number placed on the return stack by >R will be used by ; as a return address, with fatal results. On the other hand,

```
: HARMLESS   0 >R  BEGIN  1  UNTIL  R> DROP ;
```

is indeed harmless.

Sometimes it's a little difficult to remember which of the two, >R or R>, transfers data **to** the return stack. Forth attempts to keep frequently used words short to avoid lengthy manuscripts. The pictorial value of these two words is intended to suggest moving data onto (>R) or off of (R>) the return stack.

You should develop the habit of referring to the MMSFORTH Glossary (Appendix A9, available to licensed users) when you need a reminder of the definition of any MMSFORTH word.

It is worthwhile to pause here long enough to work out what will happen if you try to use a formation like:

```
... >R  phrase   R@ phrase  R> ...
```

within a definition. Since >R moves the top parameter stack item to the return stack and R@ **copies** the item back onto the parameter stack, this construct results in leaving the parameter stack with a duplicate set of what was formerly its top item. This is a useful way to get a temporary "constant".

EXERCISE

Use >R and R> to define **2SWAP**, to swap the first two byte pairs on the stack with the third and fourth pairs. That is, after:

```
1 2 3 4 5  2SWAP
```

the stack should contain:

```
1 4 5 2 3  (with 3 on the top).
```

6.5 CONTROLLED LOOPS

MMSFORTH, like many other versions of Forth, uses the Return Stack to store the loop index (**I**) and the limit values used in controlled loops. So in MMSFORTH, I is equivalent to R@.

An alternate definition for TEN-TIMES (from Section 6.3) makes use of the DO ... LOOP construct:

```
: TEN-TIMES   10 0 DO  phrase  LOOP ;
```

where:

```
: TEN-TIMES
     10 0          Places the loop parameters on the stack.
     DO            Transfers the loop parameters to the
                   return stack.
        phrase
     LOOP ;        Repeats the loop ten times.
```

The first two numbers are, as always, placed on the stack (first 10, then 0). The 10 becomes the **limit** of the DO ... LOOP; the 0 becomes the initial value of **I**, the loop **index.** The loop will execute ten times with the index starting at 0. The DO word causes the two top words on the stack to be transferred over to the return stack; that gets the loop parameters off of the parameter stack. After the body of the loop is executed, LOOP will increment the index and compare it with the limit. If the index is less than the limit, the loop is repeated. If the loop limit has been reached or exceeded, the two values are removed from the return stack and the next word (following LOOP) is executed. Because the test comes at LOOP, a DO ... LOOP will always be executed at least once.

Sometimes it is useful to have access to the loop index in a DO ... LOOP. The Forth word I, which can be thought of as Index in the DO ... LOOP construct, fetches the top of the return stack (where the loop index is stored) and **copies** it onto the parameter stack without affecting the return stack. To print out ten numbers, from 0 through 9, use this sequence:

```
: HANG-TEN   10 0 DO  I .  LOOP ;
```

where:

```
: HANG-TEN
    10 0 DO        Transfers the loop parameters to the
                   return stack.
        I          Copies the loop index (0, 1, 2, ..., 9)
                   to the parameter stack.
        .          Prints out the loop index from TOS.
    LOOP ;         Increments the index (on the return stack),
                   compares, repeats ten times.
```

Notice that DO ... LOOP structures, like those using BEGIN ... UNTIL, cannot be executed in the immediate mode; they must appear **only** in definitions of other words.

The loop index and limit don't have to be specified in the definition. They may be the result of prior computations or other stack manipulations, just as long as they are on the stack when DO is executed. Frequently, a definition that uses a loop requires the upper limit of a loop to be specified. Suppose, for example, you have a word called READ which reads a single data item from a device and stores it in the next sequential location of an array. You could then define ITEMS :

```
: ITEMS   0 DO  READ  LOOP ;
```

To read ten items, then, you would type the desired number of readings **before** typing ITEMS :

```
10 ITEMS
```

The 10 would be on the stack when ITEMS was executed and would serve as the upper limit for the loop.

It is important to remember that any loop will be executed at least one time because the increment-and-test function is at the **end** of the loop. It is not possible to execute a loop zero times, only one or more.

Here's another look at the DO ... LOOP we saw in Chapter 5. It counts between two numbers:

```
: RANGE   CR  1+ SWAP  DO  I 4 .R  LOOP ;
```

where:

```
: RANGE
    CR          Outputs a carriage return and line feed
                for a new line.
    1+          Increments the last number, to include
                it within the loop.
    SWAP DO     Arranges loop control and begins the loop.
      I         Copies the loop index to the parameter stack.
      4 .R      Prints TOS, right-justified in a field 4 bytes
                wide.
    LOOP ;      Repeats the loop until done.
```

Then the command,

```
4 40 RANGE
```

will display a count from 4 through 40, neatly displayed in uniform columns across the line. This example illustrates the useful phrase 1+ SWAP, used to convert an inclusive range of numbers in increasing order to the parameters expected by DO. (The phrase OVER + SWAP can be used in a similar manner to convert a start and count into parameters for DO.)

Another word used to conclude loops begun with DO is **+LOOP.** +LOOP expects a number on the stack, which it adds to the loop index before comparing the index and limit. For example, a word called EVEN may use +LOOP to print even integers ranging from zero to a specified limit:

```
: EVEN   0 DO  I .  2 +LOOP ;
```

The value 2 placed on the stack before +LOOP is used as the increment to the index, so that the index steps through successive even values. In MMSFORTH, +LOOP will accept negative increments and can count across the zero "boundary". more complicated uses of +LOOP involve computing the increment for +LOOP n the body of the loop.

In use, EVEN produces the following result:

```
10 EVEN 0 2 4 6 8  ok
```

EXERCISES

1. Define **SUM** to add the contents of a single-precision array, given its starting address and length on the stack.

2. Define **POWER** so that m n POWER computes the n-th power of m, for non-negative n.

6.6 NESTING STRUCTURES

DO ... LOOP and IF ... ELSE ... THEN sequences may contain other such sequences but only if they are properly nested. That is, one entire DO ... LOOP pair may be inside another pair but they may not overlap. The following loops, printed vertically for clarity, print out number pairs in the order (1 1), (1 2), (1 3), ..., (5 3), (5 4), (5 5):

```
: PAIRS
 6 1 DO            Counts from one to five (major loop).
     I             Fetches the major count value.
   6 1 DO          Counts from one to five (minor loop).
       DUP .       Duplicates and prints the major loop.
       SPACE       Separates the two pairs with an extra space.
       I .         Prints the minor loop.
       CR          Types a carriage return.
     LOOP
     DROP          Discards the old major count value.
   LOOP ;
```

This definition of PAIRS has one DO ... LOOP construction nested within another. This brings us to the second nesting rule:

> **RULE 9: WHEN NESTING STRUCTURES IN FORTH, YOU MUST NEST EACH STRUCTURE COMPLETELY WITHIN ANY OUTER STRUCTURE.**

For example, you may not use IF to branch into or out of a loop or another conditional.

Another example of nesting is provided by a different definition of EVEN (Section 6.5). This EVEN performs as the first did but in addition assures that the limit is not zero:

```
: EVEN   ?DUP  IF  0 DO  I .  2 +LOOP  THEN ;
```

where:

```
: EVEN
   ?DUP IF         Checks for a non-zero limit.
   0 DO
     I .           Prints the even index value.
     2 +LOOP       Increments the loop counter by two.
   THEN ;
```

The complexity of the actions taken on either branch of an IF ... ELSE ... THEN or within a DO ... LOOP structure is virtually unlimited. For example, a complete IF ... ELSE ... THEN structure may be used within an IF branch as in this definition of NEXT, which stores a number into the next empty one of three locations, given the number and the first address on the stack:

```
: NEXT
    DUP @           Fetches the contents of the first location.
    IF              Tests it for zero.
       1+ DUP @     Fetches the contents of the next location.
       IF           Tests for zero.
          1+        Increments to the last location.
       THEN
    THEN
    ! ;             Stores the number in the first,
                    second, or third location.
```

Here we have nested one IF ... THEN structure entirely within another. This conforms to Rule 9 given above: when nesting structures in Forth, you must nest each structure completely within any outer structure.

EXERCISES

1. How would you define MAX, MIN, and ABS? (All are supplied as part of standard Forth.)

2. Define FACTORIAL to compute the factorial of a number.

6.7 RECAPITULATION

The words IF ... ELSE ... THEN, DO ... LOOP, DO ... +LOOP, and BEGIN ... UNTIL, are all compiling words (like the more advanced ones listed at the beginning of this Chapter). That is, they direct Forth's compiler to build branches within a definition, which will later cause its interpreter to re-execute or skip over words in the definition when the defined word is actually invoked. It is the function of these words to place items in a definition. Therefore you must conform to Rule 6: **compiling words must never be used outside a definition.**

The least that can happen by ignoring this rule is that trash will be left at the top of the dictionary or stack. The worst is that trash may be deposited into existing definitions, making them unusable.

7.0 PROGRAM DEVELOPMENT

In this chapter, we will create several complete Forth programs instead of minor routines. (At last!) But first, we will discuss the philosophy Forth imposes on good programming.

7.1 PROGRAMMING PHILOSOPHY

Understanding the problem you are trying to solve is essential to writing a successful application. Otherwise, sitting down at the keyboard is like driving a great distance without a road map - possible, but not efficient. In order to uncover the essential features of the desired application, long hours of study may be required. Some tentative write/reuse tradeoffs must be made. Initial decisions need to be made about how much existing code will be applicable and how much to develop afresh. In common business applications, for example, standard packages may be adopted but a lot of custom requirements will be best handled in a complete design. If the computer must contact the outside world, appropriate hardware/software tradeoffs must be considered as well. Once the requirements are specified and the point of departure is selected, application design and implementation begins.

There are two schools of thought in programming methodology. The first holds that you should write your program at your desk, check it carefully, and then use your terminal to implement and test it. The other school would prefer that you write the program at the terminal, while you're on-line with the development system. You may choose either (or both), depending upon your skills and willingness to experiment. Eventually, however, you will need to load in the MMSFORTH System and start operating.

When the initial application words have been tested, you will probably edit the source text onto the diskette using the Editor (discussed in Chapter 2). An application name, such as PROJECT, can be associated with source text by editing a constant or a colon definition into the diskette directory. For example,

```
: PROJECT   80 5 LOADS ;
```

associates Block 80 with PROJECT; 80 is the first load screen number of five for the application (which may, in turn, load other screens). Typing PROJECT will then compile your application into memory for testing.

Your MMSFORTH System supports interactive program development. When your application is completely debugged, the interactive features, such as the Editor and the source blocks Search utility, may be no longer needed. You can delete appropriate sections of code, and reload. At this point you can also edit with neat commenting on the source blocks, can shift block allocations for best final configuration, and can merge your

application blocks with a client or customer's MMSFORTH System. Finally, you may opt to use MMSFORTH's CUSTOMIZE Utility to reprecompile the merged code for rapid system start-up, minimum disk usage, and/or protection of your source code.

7.2 TOP-DOWN DESIGN

Frequently you can sketch out an application in reverse. That is, you begin by writing tentative definitions of words to perform the major functions of your application. The effort of drafting these definitions will clarify for you which other words need to be written first in order for the major words to work. In this way you continue backwards until you determine what variables, constants, and basic "building block" definitions you will need. Of course, to be loaded your application will have to be entered, starting with the simplest definitions and building up to the most complex. But the top-down approach will enable you to identify the most convenient elements to define. The second example presented in this chapter will follow the top-down approach.

7.3 TESTING AND DEBUGGING

You will seldom write a complete application at once. It is better to write and debug a screen or two of source text at a time. You can compile your new definitions and test them out individually at the keyboard, checking that they maintain the stack as you desire and otherwise perform correctly. This facilitates debugging by breaking the application up into easily tested modules.

A technique that is frequently helpful in testing a partially developed application is the use of **stubs.** Stubs are short Forth definitions used temporarily to define words that the software under test will invoke. Stubs can be used to simulate the behavior of hardware which isn't currently connected. Another use that arises with the top-down approach is the simulation of a basic word before its final definition is written, in order to debug a word at a higher level.

7.4 **A BEGINNER'S EXERCISE** ("PAINT" PROGRAM)

This first program has not been built in the top-down manner. Instead, it is a step-by-step trip into Forth, suitable for initial experimentation before wading through the deepest secrets of this MMSFORTH USERS MANUAL and GLOSSARY. In fact, it makes an excellent demonstration to your co-workers or computer club. Many will recognize it as the same "white-out" operation demonstrated in the Radio Shack Editor/Assembler Manual, but with a vast improvement in the ease of implementation!

We are going to create a new routine based on the Forth word, **FILL.** We will do so by using some immediate-execute mode operations. We also will store the program in an edited block for further modification later.

First, try the word FILL to see how it performs. In your MMSFORTH Glossary we find that FILL expects three numbers on stack and will fill the 2OS number of bytes starting at the address 3OS, with the character whose ASCII code is TOS. On TRS-80, use it to paint the 1024-character screen (starting at RAM location 15360) white with ASCII Code 191 :

```
15360 1024 191 FILL  (Enter)
```

> Note: The IBM Personal Computer varies considerably on this example. Appropriate source code is included on an upper block of the IBM System Disk.

Fast, what? But we can make it faster by defining a single new Forth word to do all except the ASCII code, and then we can use various codes to get different displays. We will have to get the ASCII code number from below the others on stack (3OS) and rotate it to the top with a ROT immediately before calling FILL. Before we start, put a special word at the beginning of your new words, like a bookmark:

```
: TASK ;
: PAINT  15360 1024 ROT FILL ;
191 PAINT  Enter
```

Get the idea? If you believe that, we'll tell you another:

```
: WHITE      191 PAINT ;
: BLACK      128 PAINT ;
: STRIPE     131 PAINT ;
WHITE  Enter    BLACK  Enter    STRIPE  Enter
```

Wow! We can even make it flash, like so:

```
: FLASHES   0 DO  WHITE  BLACK  LOOP ;

5 FLASHES  Enter
```

Hmm, a problem! Forth is so fast that the flashes are ineffective. We can add a pause routine, but first we will forget the latest word so we can redefine it with the new PAUSE word pre-defined for use therein:

```
FORGET FLASHES
: PAUSE   1000 0 DO  LOOP ;
: FLASHES   0 DO  WHITE PAUSE  BLACK PAUSE PAUSE  LOOP ;
5 FLASHES
```

Why the double pause after BLACK ? Because it looked better, that's why. Try it yourself. When you've had enough, you can remove these new words from the top of the dictionary and free the RAM for other uses, by saying FORGET TASK. Then use these techniques and your Glossary for other projects.

Now to contemplate that which we have created. Using Forth, we quickly and easily added special new words for our own tasks to the existing FORTH vocabulary of the MMSFORTH dictionary. These words are compiled immediately; that is, their machine code definitions are added to the dictionary in RAM and they become as useable as the preceding words. But the source code, the sequence of Forth words we wrote on the screen, is no longer available for modification or to reload tomorrow night. To store it, we need an Editor.

As fate would have it, we have a fine Full-screen Editor aboard on our MMSFORTH system tape or disk. It can be used to create the source code in a Forth block, from which it can be used, modified and stored to tape or disk.

The MMSFORTH block is the same size as the TRS-80 video display screen: 16 lines (numbered 0 through 15) of 64 characters each, adding up to 1024 characters or bytes or, as we say in the byte business, 1K. Your original MMSFORTH System has over 100 blocks of software aboard; on the Model I 35-track disk, even with both a system and a programs diskette, there is only one additional blank block on the system disk. For now, we will build the program in a temporary block, in the computer's memory. Two block buffers are provided (and there are options for more), and the actual block numbers presently "attached" to their contents are up to you and your Editor's COPY function. Taking care to select an available block number (we'll use 86 in this example), enter the Editor vocabulary on that block:

```
86 EDIT
```

and clear it by deleting all its lines: hold down the Alternate key (Shift and Clear keys on the TRS-80) while pressing the D key to Delete lines with the keyboard auto-repeat function.

Now you have a clean slate on which to create the source code record. Start by titling your work of art on Line 0. You are already in the Replace mode, so write:

```
( MY FIRST MASTERPIECE! )
```

or some other significant statement. Add **: TASK ;** near the right end of this line. In doing so, be sure to leave the last column unoccupied. That's a good general rule, to avoid load problems resulting from wraparound (one line running into the next). Then press Enter to move down to the beginning of the next line. Let's skip a line with another Enter, just to keep things readable and to leave some room for possible future changes. On Line 2 (remember, we started on Line 0), we again can enter:

```
: PAINT   15360 1024 ROT FILL ;
```

We are still in the Editor's default Replace submode (you can tell by the shape of the cursor character). Two of the other submodes are Insert (hold the Insert key or Clear and I together to enter it, and the same when you wish to return to the Replace submode), and Delete (tap the Delete key on IBM or on TRS-80, hold Clear and tap D) to gobble one character from the right, or continue holding to gobble many characters. (We will refer to the TRS-80's Clear key as a "control key" from now on.) The Editor's Insert and Delete functions will come in handy if you make a typographical error. Experiment, then complete the earlier program on this block. The Editor has many more tricks up its sleeve, but these submodes are all you'll need to correct mistakes and to make changes.

When you get the block the way you like, be sure to save it. From the Editor mode press Alternate S (TRS-80's shift-control-S) to **Substitute** this version for the prior one in the block buffer (or press Alternate-Q to **Quit** it, keeping the prior one). Now it is the new Block 86, or whatever (you can display the present TRS-80 Block number with Alternate-B). You can again compile from this source code, by entering **86 LOAD.** This time the source code will still be available in Block 86 when you are through. Try it out, by entering **5 FLASHES** again. But remember that repeated LOADs will compile more and more copies of your source code into the limited amount of RAM and eventually may crash the system; then you will have to reload Forth or, if you are on a TRS-80, you might reenter it from the ROM BASIC by pressing Break, Reset, and SYSTEM Enter /19200. We think you will agree that it is safer and easier to **FORGET TASK** (the first new word) before reLOADing!

As you get into fancier programming, you will create multiple-block programs and Forth's virtual memory will move the modified earlier blocks out to disk or tape as they are displaced by new text entering the block buffers. But for now, and for the final two blocks later, if you want to keep this block move it out to tape or disk yourself with the word, FLUSH. If you'd rather forget the present contents of the block buffers, key in EMPTY-BUFFERS to ensure that the virtual memory doesn't flush them for you later. To FLUSH to tape, you must locate the correct place on tape, in Record mode. The disk system overwrites that block number automatically. Be sure you aren't write-protected or PBLK'd (PBLK is MMSFORTH's software write-protect variable, which is set to the lowest unprotected block number).

Now you have created your own Forth program, saved its source in a Forth block, and saved that block for use tomorrow. Congratulations, you're ready to GO FORTH!

7.5 **AN INTERMEDIATE APPLICATION** ("SIMPLE SIMON" PROGRAM)

We have chosen the implementation of a simple graphics game as a useful and interesting example. The purpose is not to elaborate any particular application but rather to illustrate the programming techniques and development cycle discussed at the beginning of this chapter. Additionally, it uses some rather interesting methods to generate sounds, to scan the keyboard and simulate it on screen, and to output to a port. (On TRS-80, 255 OUT goes to the cassette recorder or a speaker, etc.). Those interested in process control applications will recognize the specific usefulness of these routines.

SIMPLE SIMON uses the computer hardware (plus an optional speaker on TRS-80) to emulate some of the features of a popular children's electronic game. Instead of the standard game's four numbered buttons we will scan and use the nine main digits on the numeric keypad. We will simulate this simplified keypad on screen, flashing a block of light on the appropriate button when it is "pushed" by the computer or when the real one is pushed by you. The computer will test your ability to follow a specific sequence of buttons, adding one at a time until you miss. Then it will blink your wrong choice repeatedly, tell you how many you did successfully, and query whether another game will be played. Sound routines are also provided, synchronized with this action. On the TRS-80, the sound output can be monitored through your cassette recorder or a speaker/amplifier, in the manner described for the MMSFORTH System's BREAKFORTH game.

We recommend that you study these TRS-80 blocks (or the SIMON Program on the IBM disk) to see what they are about, referring to the following comments on the more interesting routines. Not all operations are explained in detail, but the clues are here and you will be able to work out the rest step by step as you progress through the book and your own learning curve. Meanwhile, use MMSFORTH's Editor or THE NOTEPAD to transcribe these blocks onto your own TRS-80 MMSFORTH System (in any three consecutive blocks) for further study and for the fun of running the program.

7.5.1 Listing of "Simple Simon" Program

```
Block: 132
 0  ( 06/19/81 SIMON 1 of 3, by David Huntress; MMS mods for V2.0 )
 1  : TASK ;   DIRBLK 13 + LOAD ( random numbers )
 2  CREATE COMPKEYS 33 ALLOT   143 CONSTANT WHITE    VARIABLE TIME
 3
 4  : TO-SQUARE ( key# ASCII-char -> , set cursor & output char )
 5    SWAP 1- 3 /MOD  7 SWAP -  SWAP 4 * 26 +  PTC   EMIT ;
 6
 7  : PAUSE ( delay-count -> )  0 DO LOOP ;
 8
 9  : TONE ( 1/2-cycle-delay-count  #cycles -> , via cassette port )
10    0 DO 9 255 OUTP DUP PAUSE  8 255 OUTP DUP PAUSE LOOP DROP ;
11
12  ( CODE :DI  DI NEXT   CODE :EI  EI NEXT  ( disables interrupts )
13  ( : TONE  :DI TONE :EI ;                  (  for "nicer" tones  )
14
15  : KEYTONE ( key# -> ) TIME @ 2* OVER 10 + / TONE ;          -->

Block: 133
 0  ( 06/19/81 SIMON 2 of 3, by David Huntress; MMS mods for V2.0 )
 1
 2  : KEYNUMBER ( key# -> key# ASCII-key# ) DUP 48 + ;
 3
 4  : ISRIGHT  ( key# -> )  TIME @ PAUSE  DUP WHITE TO-SQUARE
 5    DUP KEYTONE  KEYNUMBER TO-SQUARE ;
 6
 7  : ISWRONG  ( key# -> )  6 0 DO  DUP WHITE TO-SQUARE  10 20 TONE
 8    DUP KEYNUMBER TO-SQUARE  12 20 TONE  LOOP  DROP ;
 9
10  : COMPKEY ( round# -> key# ) COMPKEYS + C@ ;
11
12  : COMPMOVE ( round# ->) 600 TIME ! 0 DO I COMPKEY ISRIGHT LOOP ;
13
14   : YOURKEY ( -> key# , input a number from 1 to 9 )
15    BEGIN ?KEY 48 - DUP 1 < OVER 9 > OR  WHILE DROP  REPEAT ; -->

Block: 134
 0  ( 06/19/81 SIMON 3 of 3, by David Huntress; MMS mods for V2.0 )
 1
 2  : YOURMOVE ( round# -> err-flag ) 300 TIME ! 0 SWAP 0 DO YOURKEY
 3    DUP I COMPKEY = IF ISRIGHT ELSE ISWRONG 1+ LEAVE THEN LOOP ;
 4
 5  : SETCOMPKEYS RANDOMIZE  33 0 DO  9 RND  COMPKEYS I + C! LOOP ;
 6
 7  : DRAWBOARD  PAGE 23 EMIT 3 8 PTC ." S I M P L E   S I M O N"
 8    10 1 DO  I KEYNUMBER TO-SQUARE  LOOP ;
 9
10  : SIMON ( , have cassette hooked up for sound )
11    BEGIN   DRAWBOARD  SETCOMPKEYS      0 ( round# )
12      BEGIN 7000 PAUSE 1+ DUP COMPMOVE DUP YOURMOVE OVER 32 = OR
13      UNTIL 11 12 PTC ." You got to Round " .
14            12 12 PTC ." Try again" Y/N
15    UNTIL ;                      SIMON  FORGET TASK  PROGRAMS
```

7.5.2 **Analysis of "Simple Simon" Program**

Because this Forth program has its highest level operations defined last, building upon the prior definitions, we will read it from the back, or "the main act".

Block 134:

Following the final definition, : SIMON, the immediate word SIMON executes the progam. SIMON's outer BEGIN ... UNTIL loop continues indefinitely for as many game rounds as are answered with a "Y" to the "Try again?" query. Then control passes to the FORGET TASK phrase which removes all SIMON code (including the RANDOM numbers extension which it caused to be loaded) and to the word PROGRAMS which returns MMSFORTH's Programs Menu to the screen.

: SIMON

Begins by drawing the three-by-three numbered board, then sets a random key sequence for the computer to divulge, one new key at a time. Just before entering the inner of its two nested BEGIN ... UNTIL loops, it zeroes a move counter on the stack; this will get a 1+ on each pass through the inner loop which plays a round. After the losing round, the second half of the outer loop displays that score and asks if you want to play again. A "Y" answer puts a zero on stack for the final UNTIL, returning to BEGIN, or a "N" ends the program. The inner loop pauses, makes the computer's move, awaits and then displays the player's move, compares and continues if the string of moves agrees.

: DRAWBOARD

Clears the screen, sets the double-width character display mode, titles the screen and then uses a DO ... LOOP structure and the KEYNUMBER and TO-SQUARE words to write each of the 9 keys to the proper place on screen.

: SETCOMPKEYS

Randomly fills 33 bytes, the computer key sequence, with values from 1 to 9 into the COMPKEYS array.

: YOURMOVE

Resets the TIME delay for the PAUSE routine, compares your latest move to the appropriate one in the COMPKEY array, and proceeds or complains appropriately.

Block 133:

: YOURKEY
Accepts your key entry if it is between 1 and 9.

: COMPMOVE
Resets the TIME delay for PAUSE, outputs the next computer key, signals that it is correct and proceeds.

: COMPKEY
Gets the next value in the sequence from the COMPKEYS array.

: ISWRONG
Alternates a white flash with the last-input key number, while beeping the sound quickly to signal an error.

: ISRIGHT
Flashes the right key more slowly, with sound to match.

: KEYNUMBER
Makes a copy of the keynumber and adds 48 decimal to it to create the ASCII code which will display it.

Block 132:

: KEYTONE
Adjusts the tone frequency and number of pulses to normalize the over-all time of the sound.

CODE :DI
This CODE sequence creates a high-level Forth word to temporarily disable interrupts. It may be elected by removing the two left-margin open-parentheses, for a purer sound on Model III TRS-80's.

CODE :EI
Used with :DI.

: TONE
Creates a square-wave sound signal by outputing the TOS number of 8-pulses and 9-pulses to Port 255, the TRS-80's cassette port.

: PAUSE
Is a simple delay, taking its time from TOS.

: TO-SQUARE
This elegant, specialized routine moves the cursor to the appropriate spot for any of the 9 keys, then EMITs its character onto that spot.

TIME
This variable is automatically initialized to 0.

WHITE
EMITting this constant's ASCII code creates the white flash.

CREATE COMPKEYS 33 ALLOT
Labels and reserves space for the 33-byte COMPKEYS array.

DIRBLK 13 + LOAD
Does a relative load of the Random numbers block, so we can use RANDOMIZE in Block 134's SETCOMPKEYS routine.

: TASK ;
"Bookmarks" the beginning of our task so we can FORGET it later.

8.0 **ADVANCED PROGRAM DEVELOPMENT** (CHECKBOOK PROGRAM)

8.1 CHAPTER OUTLINE

This chapter studies the CHECKBOOK program which is delivered with the MMSFORTH System. This demonstration business program provides an excellent example for advanced Forth programmers. The chapter consists of three parts:

1. A brief description of purpose for the CHECKBOOK program, including a data file layout.

2. A listing of its six source blocks.

3. A detailed explanation, word by word, of the program's Forth source code.

The latter starts at the highest level of the program, which is defined last in the six source blocks, and works back through the blocks to the lowest level words. This "top-down design, bottom-up coding" approach should be used to create **your** Forth programs, as well as to analyze others as we do here.

8.2 PURPOSE

The CHECKBOOK program is designed to provide simple checkbook balancing and reporting capabilities. It implements a simple file structure, and is menu driven for ease of use. However, note that it is **not** a complete, well-implemented business package typical of those professionally designed by MMS or other custom software houses. In fact, it took less than a day to design, write and debug this example.

As an advanced-level demonstration, the CHECKBOOK program exercises Forth techniques for text display, for double-precision integer mathematics and simple scaling techniques, for formatted numeric display in dollars and cents, and for one-touch menu selection using both ACASE and NCASE, MMSFORTH's alphanumeric and numeric CASE statements. Its file structure provides a model for many other applications.

8.3 LISTING OF "CHECKBOOK" PROGRAM

```
Block: 112
 0  ( 06/28/81 CHECKBOOK, 1 of 6, plus 2 data blocks at end )
 1  ( FORGET SCR   : DIR  BOOT ;  ( for 16K Systems )  : TASK ;
 2    DIRBLK  3 + LOAD ( Double numbers ) FORGET T1
 3    DIRBLK 10 + LOAD ( strings )        FORGET RIGHT$
 4  VARIABLE IS  VARIABLE IC  2VARIABLE IA   23 $VARIABLE IP$
 5  2VARIABLE BALANCE  0 CONSTANT #CKS  BLK @ 6 + CONSTANT CHKDATA
 6  : ABAL CHKDATA BLOCK ;           : A#CKS CHKDATA 1+ BLOCK ;
 7  : REC# 0 34 U/MOD CHKDATA + BLOCK SWAP 30 * + 4 + ;
 8  : CK# REC# ;     : AMOUNT REC# 2+ ;       : NAME REC# 6 + ;
 9  : 999-OUT BEGIN 2DUP 1000. D- SWAP DROP 0< IF #S 1 ELSE
10    # # # 44 HOLD 0 THEN UNTIL ;
11  : $, SWAP OVER DABS <# # # 46 HOLD 999-OUT 36 HOLD ROT
12    SIGN BL HOLD #> ;                        : .$$ $, TYPE ;
13  : B-OUT 2 33 PTC ." Current balance" BALANCE 2@ .$$ ;
14  : FIND? 0 SWAP #CKS IF #CKS 0 DO DUP I CK# @ = IF SWAP 1+
15    SWAP I IS ! LEAVE THEN LOOP THEN DROP ;                  -->

Block: 113
 0  ( 06/28/81 CHECKBOOK, 2 of 6 )
 1  : ??DUP DUP FIND? IF DROP 0 1 ELSE 0 THEN ;
 2  : GET-CK# BEGIN CR ." Enter check number" #IN ??DUP IF CR
 3    ." That check number already on file" THEN ?DUP UNTIL IC ! ;
 4  : GET-PAYEE CR ." Enter payee name" IN$ 23 LEFT$ IP$ $! ;
 5  : 10.* 2DUP 2DUP D+ 2DUP D+ D+ 2DUP D+ ;
 6  : $#IN BEGIN ."  ? " PAD 15 EXPECT PAD 1 - NUMBER DROP
 7    #PT C@ ?DUP IF DUP 1 = IF DROP 10.* 10.* 1
 8    ELSE 3 = IF 1 ELSE 2DROP 0  CR ." Redo"
 9    THEN THEN ELSE HI# @ 10.* 10.* 1 THEN UNTIL ;
10  : GET-AMOUNT BEGIN CR ." Enter check amount as a positive number
11  " $#IN DUP 0< IF 2DROP 0 ELSE 1 THEN UNTIL IA 2! ;          -->
12
13
14
15

Block: 114
 0  ( 06/28/81 CHECKBOOK, 3 of 6 )
 1  : VERIFY BEGIN PAGE CR CR ." 1. Check number  " IC @ . CR
 2                            ." 2. Amount        " IA 2@ .$$ CR
 3                            ." 3. Payee         " IP$ $. CR
 4    CR ." If everything is correct press (Enter)"
 5    CR ." Otherwise type the line number of the data in error ? "
 6    BEGIN 1 KEY DUP EMIT
 7      NCASE 13 49 50 51 " 1+ GET-CK# GET-AMOUNT GET-PAYEE
 8      OTHERWISE CR ." Invalid line number, reenter ? " 1- CASEND
 9    ?DUP UNTIL 1- UNTIL ;
10  : CHECKS CR BEGIN ." Enter more checks" Y/N 0= WHILE
11    PAGE GET-CK#   GET-AMOUNT  GET-PAYEE  VERIFY
12    IC @ #CKS CK# ! IA 2@ #CKS AMOUNT 2! IP$ #CKS NAME $!
13    UPDATE 1 ' #CKS +! BALANCE 2@ IA 2@ D- BALANCE 2! REPEAT ;
14  : ENTER-CHECKS PAGE ." Enter check(s) written this month"
15    CHECKS ;                                                 -->
```

```
Block: 115
 0  ( 01/18/82 CHECKBOOK, 4 of 6 )
 1  : .$$R >R $, R> OVER - DUP 0 > IF SPACES ELSE DROP THEN  TYPE ;
 2  : LCKS  PAGE ." List of outstanding checks" CR ;
 3  : CONTINUE ." To continue, press" ENTER ;
 4  : CK-PAGE 14 /MOD IF 0= IF CONTINUE LCKS THEN 0 THEN DROP ;
 5  : OUTSTANDING  LCKS #CKS IF #CKS 0 DO I CK-PAGE I CK# @ 4 .R
 6   I AMOUNT 2@ 15 .$$R 5 SPACES I NAME $. CR LOOP THEN CONTINUE ;
 7  : G+AMOUNT PAGE BEGIN BEGIN CR DUP $. ."  as a positive number"
 8    CR $#IN DUP 0< IF 2DROP 0 ELSE 1 THEN UNTIL
 9    CR ." Press (Enter) if correct, else type 'X' ? " KEY
10    DUP EMIT CR 13 = IF ROT DROP 1 ELSE 2DROP 0 THEN UNTIL ;
11  : DEPOSITS PAGE BEGIN ." Enter deposit" Y/N 0=
12    WHILE BALANCE 2@ $" Enter deposit or interest amount"
13    G+AMOUNT D+ BALANCE 2! REPEAT ;
14  : SERVICE BALANCE 2@ $" Enter service charge" G+AMOUNT
15    D- BALANCE 2! ;                                          -->

Block: 116
 0  ( 06/28/81 CHECKBOOK, 5 of 6 )
 1  : CLOSE-UP #CKS IS @ DO I 1+ CK# I CK# 30 CMOVE UPDATE LOOP
 2    -1 ' #CKS +! ;
 3  : CANCEL PAGE ." Enter number of check to be cancelled" #IN
 4    FIND? IF BALANCE 2@ IS @ AMOUNT 2@ D+ BALANCE 2! CLOSE-UP
 5          THEN ;
 6  : PREV-CHECKS PAGE ABAL 2@ BALANCE 2! A#CKS @ ' #CKS ! ;
 7  : CLEAR-CHECKS BEGIN PAGE ." Want to enter a cleared check"
 8    Y/N 0= WHILE BEGIN CR ." Enter number of cleared check"
 9    #IN FIND? IF CLOSE-UP 1 ELSE CR
10   ." That check number not found; reenter" 0 THEN UNTIL REPEAT ;
11  : INITIAL-SETUP 0 ' #CKS ! $" Enter current balance"
12    G+AMOUNT BALANCE 2! CR ." Enter outstanding check(s)" Y/N
13    0= IF CHECKS THEN ;
14  : CKEND #CKS A#CKS ! UPDATE BALANCE 2@ ABAL 2! UPDATE
15    FLUSH 1+ ;                                               -->

Block: 117
 0  ( 06/28/81 CHECKBOOK, 6 of 6 )
 1  : MENU PAGE ." MMSFORTH CHECKBOOK BALANCING DEMO" CR CR CR
 2   CR ." 1. Read in prior-session data (BEGIN session here)"
 3   CR ." 2. Enter check(s) written"
 4   CR ." 3. Enter deposit(s) or interest"
 5   CR ." 4. List outstanding check(s)"
 6   CR ." 5. Enter service charge(s)"
 7   CR ." 6. Cancel check(s)"
 8   CR ." 7. Enter check(s) cleared by bank"
 9   CR ." 8. Initialize files (BEGIN here, 1st time only!)"
10   CR ." 9. SAVE session data (END every session here)"  ;
11  : CHECKBOOK   BEGIN MENU B-OUT
12    14 0 PTC ." Enter desired option ? " 0 KEY DUP EMIT
13    ACASE 123456789" PREV-CHECKS ENTER-CHECKS DEPOSITS
14      OUTSTANDING SERVICE CANCEL CLEAR-CHECKS INITIAL-SETUP CKEND
15    CASEND UNTIL ;    CHECKBOOK   FORGET TASK   DIR
```

8.2.1 Layout of File Data Blocks

1st block, Bytes 0 - 3, ABAL = current balance.
2nd block, Bytes 0 & 1, A#CKS = number of checks stored.

All blocks, starting with Byte 4 and repeating every 30 bytes:
Bytes 4 & 5, CK# = check number of stored check.
Bytes 6 - 9, AMOUNT = amount of stored check.
Bytes 10 - 27, NAME = name of payee on stored check.

REC# - routine to calculate location of a particular check record, this routine expects record number on the stack and returns the address of the record.

```
0 34 U/MOD      Calculates relative position in block
                and relative block.
CHKDATA +       Calculates actual block number.
BLOCK           Brings that block into a block buffer and
                leaves the address of the buffer on the
                stack.
SWAP            Brings up the relative position.
30 * +          Calculates the position based on 30-byte
                records.
4 +             Adds extra to get past ABAL or A#CKS.
```

8.3 TOP-DOWN EXPLANATION OF THE PROGRAM

Block 117:

CHECKBOOK FORGET TASK DIR

Not part of the program, this immediate statement executes the program; when you are done, it forgets it and brings up the program menu (or in a 16K system where the word DIR has been redefined, it will boot the system).

: CHECKBOOK

The main program routine.

BEGIN MENU ." Enter desired option"

Starts the main loop of the program, prints the menu, and asks for your choice.

0 KEY DUP EMIT

0 sets up zero on stack for ending UNTIL. (Zero will be replaced by 1 when choice 9 of the menu is taken, thus the program will end.) KEY waits for a key to be pressed. DUP EMIT preserves the key value by duplicating it and prints the extra to screen.

ACASE 123456789" ..... CASEND UNTIL ;

ACASE is MMSFORTH's alphanumeric CASE statement. If the key pressed corresponds to one of the characters between ACASE and " , the word between " and CASEND which corresponds in location to the character is executed. If there is no match, execution continues after CASEND where UNTIL finds a zero on stack and returns to BEGIN to ask again.

: MENU ..... ;

Clears the screen and displays the menu of operations.

Block 116:

: CKEND
End-of-program routine.

#CKS A#CKS ! UPDATE
Stores number of checks from named area into first data block positions 0 - 1 (A#CKS) and marks the block as UPDATEd.

BALANCE 2@ ABAL 2! UPDATE
Stores balance into 2nd data block positions 0 - 3 (ABAL) and marks the block as Updated.

FLUSH 1+ ;
Forces the updated blocks to disk and adds a 1 to the zero left on the stack to cause UNTIL in CHECK to exit.

: INITIAL-SETUP
Routine to enter initial outstanding checks and current balance.

0 ' #CKS !
Sets initial number of checks to zero.

." Enter outstanding check(s)" Y/N
Ask for any initial outstanding checks.

0= IF CHECKS THEN
Reversed "truth value" from Y/N and if reply was Y , executes the routine to enter outstanding checks.

$. Enter current balance"
Puts message into PAD and puts the PAD address on stack for later use by G+AMOUNT routine.

G+AMOUNT BALANCE 2! ;
Gets double-precision starting balance and puts it into the variable named BALANCE .

: CLEAR-CHECKS
Routine to remove from file, those checks which have been cleared by your bank; i.e., deletes records but does not change the balance. This routine contains two loops. The outer asks for the checks, the inner makes sure the check is in the file and, if it is, deletes it and closes up the space where it was.

BEGIN PAGE ." Want to enter a cleared check" Y/N 0= WHILE
Starts the outer loop, clears the screen, and gives you the opportunity to get out if you got in by mistake, or when you are finished. Sets up truth value for WHILE (of BEGIN ... WHILE ... REPEAT loop.)

BEGIN CR ." Enter number of cleared check" #IN
Starts inner (BEGIN ... UNTIL) loop. Requests the number of the check to be cleared.

FIND?
Searches through the checks in the file for a check with the correct number. Leaves a 1 on the stack if found, else leaves a zero.

IF CLOSE-UP 1
If the check is found, closes up the file over it and leaves 1 on the stack to get out of the inner loop.

ELSE CR ." That check number not found; reenter" 0 THEN
If the check is not found, asks you to reenter and leaves 0 on the stack to repeat inner loop.

UNTIL REPEAT ;
Finishes the inner and outer loops.

: PREV-CHECKS
Routine to read in the balance and number of checks stored on the disk.

PAGE ABAL 2@ BALANCE 2!
Clears the screen and moves the balance from the disk into the variable, BALANCE .

A#CKS @ ' #CKS !
Moves the count of the checks from disk to the constant, #CKS .

: CANCEL
Routine to delete a check from the file and adjust the balance by the amount of the check.

PAGE ". Enter number of check to be cancelled" #IN
Clears the screen and requests the number of the check to be cancelled.

FIND?
Finds the check.

IF BALANCE 2@ IS @ AMOUNT 2@ D+
: If found, picks up old balance, picks up relative location of check, uses it to pick up check amount, and adds the check amount to the old balance.

BALANCE 2! CLOSE-UP THEN ;
: Stores adjusted balance, close up file over check just canceled, and finishes IF construct.

: CLOSE-UP
: Routine to compress the file to delete a check.

#CKS IS @
: Picks up number of checks in file and relative record to be deleted.

DO
: This loop takes each check record, one by one and moves it down one record.

I 1+ CK# I CK# 30 CMOVE UPDATE LOOP
: Gets address of check record corresponding to the loop counter plus 1 and gets the loop counter. Moves from the former to the latter for 30 bytes, marks the block for rewrite, and after doing all, ends loop.

-1 ' #CKS +! ;
: Subtracts 1 from the number of checks in the file.

Block 115:

: SERVICE
Routine to subtract service charges from balance.

BALANCE 2@
Picks up current balance.

$. Enter service charge" G+AMOUNT
Places request-for-service-charge message into PAD for use by G+AMOUNT, and calls G+AMOUNT.

D- BALANCE 2! ;
Subtracts amount of service charge from current balance and saves back into variable.

: DEPOSITS
Routine to process deposits and interest amounts.

PAGE BEGIN ." Enter deposit" Y/N 0= WHILE
Clears screen, starts BEGIN ... WHILE ... REPEAT loop, checks that you want to enter a (another) deposit.

BALANCE 2@ $" Enter deposit or interest amount" G+AMOUNT
Picks up the old balance, sets message for G+AMOUNT and calls it.

D+ BALANCE 2! REPEAT ;
Adds deposit to old balance, resaves in BALANCE, and closes loop.

: G+AMOUNT
Routine to get positive amount. Address of message is on TOS coming in, double-precision amount is TOS coming out.

PAGE BEGIN BEGIN CR DUP $. ." as a positive number" CR
Clears screen, starts outer and inner loops, duplicates address of message to preserve it, then prints held message and "positive" message.

$#IN DUP 0<
Gets amount, copies high order portion and checks sign for negative.

IF 2DROP 0
If negative, drops amount, puts zero on stack to repeat inner loop.

ELSE 1 THEN UNTIL
If positive, puts a 1 on stack to exit inner loop.

CR ." Press (Enter) if correct, else type 'X' ? "
Verification routine message.

KEY DUP EMIT CR 13 =
Waits for a key to be pressed, duplicates and emits the duplicate to screen, compares to carriage return (decimal 13).

IF ROT DROP 1
If carriage return, brings up 3rd item on stack (address of message) and drops it. Leave a 1 on stack to exit outer loop.

ELSE 2DROP 0 THEN
If not carriage return, drops double-precision number just read in, and leaves zero on stack to repeat outer loop.

UNTIL ;
Close outer loop.

: OUTSTANDING
Routine to list outstanding checks.

LCKS #CKS
Prints heading and gets current number of checks.

IF #CKS 0 DO
If number of checks is not zero, prepares a DO ... LOOP of the number of checks in the file.

I CK-PAGE
Check the count to see if a new page should be started.

I CK# @ 4 .R
Picks up check number and prints it right-justified in a field of length four.

I AMOUNT 2@ 15 .$$R 5 SPACES
Picks up the amount and prints it right-justified in a field of 15 using the special routine .$$R to put in commas and a decimal point. Prints five spaces following the number.

I NAME $. CR LOOP THEN
Prints the payee name then carriage return, ends the loop and ends the IF statement.

CONTINUE ;
Pauses and waits for Enter to continue.

: CK-PAGE
Routine to check for end of screen.

14 /MOD IF 0=
Checks that count of lines printed is evenly divisible by 14 (remainder = 0) and original number was not zero. (On IBM PC, 22 instead of 14.)

IF CONTINUE LCKS THEN
If both of the above were true, than pauses and waits for the Enter key to be pressed, and after it is pressed, starts a new screen.

0 THEN DROP
Puts zero on stack, exits inner IF. Drops zero or the number which would have been used by the inner IF. (Note: this code could have been ELSE DROP THEN .)

: CONTINUE ." To continue, press" ENTER ;
This routine prints the message and waits for Enter to be pressed.

: LCKS PAGE ." List of outstanding checks" CR ;
This routine clears the screen, and prints the heading message.

: .$$R
Routine to print formatted dollar amount, right justified. Expects length on TOS and double-precision amount 2OS, leaves nothing on stack.

>R $, R>
Puts length of print field on return stack for safe keeping, formats dollar amount leaving length of formatted data on stack, and brings length of print field back from return stack.

OVER - DUP 0 >
Copies formatted length to TOS, subtracts it from field length, DUPs it to save, and checks if difference is greater than zero.

IF SPACES
If the difference is greater than zero, prints that many extra spaces.

ELSE DROP THEN
If the difference is not greater than zero, drops the extra copy of the difference which is on TOS.

TYPE ;
TYPE the formatted amount.

Block 114:

: ENTER-CHECKS
Routine to input checks written, taking input from the keyboard and inserting it in the data file.

PAGE ." Enter check(s) written this month" CHECKS ;
Clears the screen, prints message and calls check entry routine.

: CHECKS
Routine to request check data.

CR BEGIN ." Enter more checks" Y/N 0= WHILE
Carriage return, starts BEGIN ... WHILE ... REPEAT loop, are-you-finished-entering-checks? message (or verify that this routine is where you want to be). The following routine will be repeated as long as the operator keeps answering Y for Yes.

PAGE GET-CK# GET-AMOUNT GET-PAYEE VERIFY
Clears the screen, gets the check number, amount, and payee, and verifies them.

IC @ #CKS CK# !
Gets check number just inputted from where it was stored (IC), gets current maximum number of check records, stores check number in next available location in file.

IA 2@ #CKS AMOUNT 2!
Gets double-precision amount just inputted from temporary storage, gets current maximum number of records, stores amount in next location in file.

IP$ #CKS NAME $!
Gets payee name just input from temporary storage, gets current maximum number of records, stores name in next location in file.

UPDATE
Marks block as updated for later writing to disk by virtual file system.

1 ' #CKS +!
Increments current maximum number of records by one.

BALANCE 2 IA 2 D- BALANCE 2! REPEAT ;
Picks up current balance and new check amount. Subtracts check from balance, and saves. Ends BEGIN ... WHILE ... REPEAT loop. (Go back to BEGIN .)

: VERIFY

Routine to allow operator to double check data just entered on checks.

BEGIN PAGE CR CR

Begins verification loop, clears screen, and moves down two lines.

." 1. Check number " IC @ . CR
." 2. Amount " IA 2@ .$$ CR
." 3. Payee " IP$ $. CR

Lists the check number, amount, and payee on a formatted screen, with each line numbered for easy identification.

CR ." If everything is correct press (Enter)"
CR ." Otherwise type the line number of the data in error ? "

Gives instructions to operator.

BEGIN 1 KEY DUP EMIT

Begins input loop, places a 1 on the stack for later use by the UNTIL, accepts a one-key input, duplicates it and emits one copy to the screen.

NCASE 13 49 50 51 "

Sets up MMSFORTH's numeric CASE statement to look for carriage-return, 1, 2, or 3.

1+ GET-CK# GET-AMOUNT GET-PAYEE

If carriage return, adds 1 to the 1 on stack to set up for exit from both inner and outer BEGIN ... UNTIL loops.
If 1, gets check number.
If 2, gets amount.
If 3, gets payee name.

OTHERWISE ." Invalid line number, reenter ? " 1- CASEND

If none of the above, prints error message, subtracts one from the 1 on stack to make it a zero for the UNTIL .

?DUP UNTIL

Duplicates the number on TOS if it is not zero. Ends inner loop.

1- UNTIL ;

Subtracts 1 from the number on stack. This number will be 2 if carriage return was pressed, or 1 if anything else happened.

Block 113:

: GET-AMOUNT
Routine to input amount for check.

BEGIN CR ." Check amount as a positive number"
Loops until valid amount is entered.

$#IN DUP 0<
Gets double-precision amount in, makes a copy of the upper portion to check the sign, compares that it is less than zero.

IF 2DROP 0
If less than zero, drops the double-precision amount, and places zero on the stack for the UNTIL.

ELSE 1 THEN
If not less than zero, keeps the double-precision amount on the stack and places a 1 on the stack for the UNTIL.

UNTIL IA 2! ;
When a positive amount has been input, saves it in IA.

: $#IN
Routine to read in a double-precision number, and to scale it to two decimal places. Scaling is often used on large number ranges in lieu of bulky and slower floating-point arithmetic.

BEGIN ." ? "
Starts loop and prints out question-mark to indicate ready to accept data.

PAD 15 EXPECT
Accepts fifteen characters into PAD.

PAD 1 - NUMBER DROP
Converts data at PAD from ASCII to numeric. Drops the extra address which NUMBER leaves on TOS.

#PT C@ ?DUP
Gets the location of the decimal point and duplicates it if it is non-zero. (#PT will be zero if there is no decimal point, 1 if there are no digits after the decimal point, 2 if one digit after the decimal point, etc.)

IF DUP 1 =
If there is a decimal point, checks to see if there are no digits after it.

IF DROP 10.* 10.* 1
If there are no digits after the decimal point, multiplies the raw number by 100 (by ten twice) to scale it. Leaves a 1 on the stack to exit the loop.

ELSE 3 =
If there are digits after the decimal point, checks for two digits.

IF 1
If there are two digits, puts a 1 on the stack to exit the loop.

ELSE 2DROP 0 CR ." Redo" THEN
If there is any other number of digits after the decimal point, it is an error. Drops the number and prepares to re-request it.

THEN
End prior IF.

ELSE HI# @ 10.* 10.* 1 THEN UNTIL ;
If there was no decimal point, the number went in as a single-precision number. Picks up high-order bits from HI#, and multiplies by 100 (times ten twice) to normalize. Places a 1 on the stack to exit the loop.

: 10.*
Routine to multiply a number by ten. This is a rather ingenious method to multiply with addition. It was devised because M*, MMSFORTH's single-times-double multiply, gives an undesired triple-precision result. A newer word, D*, is in the MMSFORTH double-precision word set and could be used instead.

2DUP 2DUP D+
Duplicates the number twice so there are three copies on the stack. Adds the top two. You now have the original number 2OS and twice-the-number TOS.

2DUP D+ D+
Duplicates twice the number and add it to itself (four times the number) then adds it to the original number giving five times the number.

2DUP D+ ;
Duplicates five times the number and adds it to itself (ten times the number).

```
: GET-PAYEE
```
Routine to input the payee name.

```
  CR ." Enter payee name" IN$
```
Requests and inputs payee name.

```
  23 LEFT$ IP$ $! ;
```
Truncates name to 23 characters and stores it in IP$.

```
: GET-CK#
```
Routine to input check number and verify it.

```
  BEGIN CR ." Enter check number" #IN
```
Begins input loop, requests and accepts check number.

```
  ??DUP
```
Checks if same check number already in file. Returns code to indicate status.

```
  IF CR ." That check number already on file" THEN
```
If check is already on file, sends message.

```
  ?DUP UNTIL IC ! ;
```
Duplicates stack if non-zero. If zero repeats routine, if non-zero stores check number in IC.

```
: ??DUP
```
Routine to check for a duplicate check number.

```
  DUP FIND?
```
Duplicates the check number to preserve it. Searches the file for it.

```
  IF DROP 0 1
```
If found, drops the check number and primes stack for IF and UNTIL in GET-CK# .

```
  ELSE 0 THEN ;
```
If not found, primes stack only for IF (the check number will do the work for the UNTIL).

Block 112:

: FIND?
Compare input check number with all the check numbers currently on file.

0 SWAP #CKS
Puts a zero on the stack and brings up the check number. Picks up the current number of checks in the file.

IF #CKS 0 DO
If the number of checks is not zero, sets up for a DO ... LOOP to look through all the checks.

DUP I CK# @ =
Makes a copy of the input check number, picks up the file check number pointed to by the loop index, and compares them to see if they are equal.

IF SWAP 1+ SWAP I IS ! LEAVE THEN
If a duplicate is found, brings the 0 which was put on the stack at the beginning of the routine up to the top, adds 1 to it (for compare later), puts it back down, and stores the loop counter at IS. Finally, LEAVE sets the loop counter to the max for the loop so that execution leaves the loop the next time LOOP is encountered.

LOOP THEN DROP ;
Closes the loop and the first IF, and gets rid of the extra copy of the input check number.

: B-OUT
Routine to print current balance at top of menu page.

2 33 PTC ." Current balance"
Places cursor at Column 33 of Line 2, and prints message.

BALANCE 2@ .$$;
Gets current balance and prints it.

: .$$ $, TYPE ;
Routine to TYPE formatted balance.

: $,
Routine to format balance.

SWAP OVER DABS
Inverts doubleprecision number, then copies high order portion up to top again. Makes the double-precision number unsigned. (The extra copy of the high order portion will be

used later to determine the sign.)

<# # # 46 HOLD
Starts formatted output which always works up from low-order to high-order digit. Converts bottom two digits to ASCII. Forces a decimal-point (ASCII Code 46) next to left of the two digits.

999-OUT
Formats remaining digits in groups of three digits, set apart by commas.

36 HOLD
Forces a dollar-sign (ASCII Code 36) at left side of the field.

ROT SIGN
Brings up the extra, signed, portion, pulls the sign off it, and puts it into the formatted output string to the left of the dollar sign.

BL HOLD #> ;
Forces a blank in as the rightmost character in the formatted output and closes the formatting.

: 999-OUT
Routine to format the integer portion of the dollars/cents in balance.

BEGIN 2DUP
Begins the loop to set off groups of three digits with commas. Duplicates the number.

1000. D- SWAP DROP 0<
Subtracts double-precision 1000 (designated by the inclusion of a decimal point) from the remaining double-precision amount on the stack, then drops the lower portion because all we are interested in is the sign. Checks to see if it is negative.

IF #S 1
If negative, finished with the loop; takes the remaining digits from the number, puts them in the formatted output string, and puts a 1 on the stack to get out.

ELSE # # # 44 HOLD 0 THEN
If positive, the number left is greater than 1000, so places the next three digits into the formatted output string, then places a comma (ASCII Code 44) there, and finally puts a zero on the stack to repeat the loop.

UNTIL ;
Closes loop.

: CK# REC# ;
Routine to find the check number within a record. (At the beginning of the record.)

: AMOUNT REC# 2+ ;
Routine to find the amount within a record. (At the beginning of the record plus 2.

: NAME REC# 6 + ;
Routine to find the payee name field within a record. (At the beginning of the record plus 6.)

: REC#
Routine to find the beginning of a record.

0 34 U/MOD
Forces the single-precision number on TOS to double-precision by putting zero on TOS, then does an unsigned divide by 34 leaving remainder and quotient which will be used to find the relative block position of the record.

CHKDATA + BLOCK
Picks up starting block number of file and adds relative block number to it, then gets the block.

SWAP 30 * + 4 + ;
Brings up the relative record within that block (the remainder from the divide), multiplies it by the record length, and offsets it by 4 to get past the balance field which is in the first four bytes of the block.

: A#CKS CHKDATA 1+ BLOCK ;
Defines the storage place for the number of checks in the file as the beginning of the second block. (First block plus one.)

: ABAL CHKDATA BLOCK ;
Defines the storage place for the current balance as the beginning of the first block.

BLK @ 6 + CONSTANT CHKDATA
Defines a constant which is six blocks up from the current block as the beginning of the data area.

0 CONSTANT #CKS
Defines a constant to hold the current number of checks. This is defined as a constant rather than as a variable to save time and memory, because it is called many more times than it is changed.

2VARIABLE BALANCE
Defines a double-precision variable to hold the current balance.

23 $VARIABLE IP$
Defines a string variable of length 23 to hold the payee name while it is being verified.

2VARIABLE IA
Defines a double-precision variable to hold the amount while it is being verified.

VARIABLE IC
Defines a single-precision variable to hold the check number while it is being verified.

VARIABLE IS
Defines a single-precision variable to hold the number of the current record within the file.

DIRBLK 10 + LOAD FORGET RIGHT$
Loads the STRINGS routines (from ten blocks above the directory block) and forgets extra portions that are not needed.

DIRBLK 3 + LOAD FORGET T1
Loads the double-precision number routines (from three blocks above the directory block) and forgets extra portions that are not needed.

(FORGET SCR : DIR BOOT ; (for 16K Systems) : TASK ;
Defines the beginning of the program for later FORGETting. If in a 16K system remove the (and it will forget the Editor and a bit more to free enough memory to run the program and will redefine DIR so that the system will boot when you exit the program.

A1.0 GETTING STARTED WITH MMSFORTH

The instructions in this Appendix section will assume that you are using a Version 2.0 or 2.1 MMSFORTH System Diskette.

> TRS-80 M.1: MMS delivers these MMSFORTH Systems **without** lowercase; it is added from the Option Select Block, typically Block 15 (see Section A1.2). IF YOUR MODEL I TRS-80 DOES NOT HAVE LOWER CASE CAPABILITIES, YOU MUST USE OUR **ALLCAPS** UTILITY BEFORE EDITING THE SOURCE CODE BLOCKS (see Appendix A4.6). Also, Model I MMSFORTH Systems arrive sized for 32K RAM, as are the demo programs. You can adjust them for use on your 16K Model I, by CUSTOMIZing on a Model I with at least 32K (see Section A1.2.2) or by asking MMS to rewrite your MMSFORTH System Diskette in this manner.

MMS has the following general recommendations:

1. Bring up a simple version, before attempting complex combinations.

2. BACKUP ALL IMPORTANT DISKETTES, AND DON'T REMOVE THE WRITE-PROTECT TAB ON YOUR MASTER DISKETTE unless you know what you are doing - MMS rewrites them for licensed users, at $10.00 plus shipping.

3. Remove diskettes from drives before throwing power switches on or off.

4. Let us help. Don't blame the MMSFORTH until you have read all the instructions, checked with MMS or a knowledgeable user, or tried the same action on a similar computer system, etc.

A1.1 **START-UP OPERATIONS**

Power up all appropriate switches on your computer system before installing any diskettes. Then mount your new MMSFORTH System Diskette in Drive 0 (we designate Disk Drives starting ad Drive 0) and "boot" the system by pressing the TRS-80's Reset button or the IBM PC's Control, Alternate & Delete buttons. (You can normally clear and initialize the system by rebooting in this manner, or with the MMSFORTH word BOOT. In several seconds, a precompiled version of MMSFORTH will load, "auto-command" a directory menu, and the Forth cursor will blink back at you. Reboot and, before moving on to the directory, halt the video display by pressing Shift-@ on the TRS-80 or Control-NumLock on the IBM PC as soon as the DUP NAME message appears. Record your MMSFORTH version and serial number on both copies of your User Licensing Agreement and on the first page of these instructions. Then press any regular key and the loading operation will continue. Later, entering DIR will return this directory menu to the screen.

Before experimenting further, make a working copy of your system diskette. MMSFORTH has separate FORMAT and BACKUP commands. The destination diskette must be formatted before you do a backup, but you need not bulk-erase; BACKUP will rewrite on any diskette already formatted by FORMAT, TRSDOS, NEWDOS, IBMDOS, etc. (But don't forget to remove the write-protect tab, first!)

If your destination diskette needs formatting enter FORMAT from the directory menu. Follow with BACKUP when done. (BACKUP will elect a one-drive routine when appropriate.) If yours is a Model I System, also backup a copy of the accompanying Programs Diskette.

IMPORTANT!! Now carefully store your original MMSFORTH System Diskette where it will be available for back-up "insurance" and for exchange for later versions. All other copies with or without modifications are for use on your one computer only, not for distribution to others or for use on multiple computer systems! Also complete, sign and return your MMSFORTH User Licensing Agreement - it is your passport to legitimate use, to your copy of the MMSFORTH Glossary and a sample copy of the MMSFORTH Newsletter, and to continuing support from Miller Microcomputer Services and/or your dealer.

Insert your newly backed-up MMSFORTH System Diskette in Drive 0, reboot, and try out some of the RPN math exercises in Chapter 1 at this time.

Forth shouldn't be all work and no play. Keep plugging away at the Chapters in this MANUAL, but also take breaks and try trips of your own. Start with small trips, to avoid early discouragement from problems which will seem small when you revisit them later.

For some early side trips, we have provided an interesting and useful series of applications programs on your initial diskette(s). Sneak an early look at Appendix A3 for lots of ideas in this regard. Try them all, fiddling with the easy ones but leaving the problems to get sorted out as you proceed.

We also recommend a side trip into the PAINT program example in Appendix A7. It is easy and well explained, suited for an early taste of Forth tricks. Later, remember it as a good starting point for you to show friends why they, too, might want to buy and use MMSFORTH.

Be sure to take time to stare at the Table of Contents so you will flip to the right place when a question occurs, to read the Preface, and to pay plenty of attention to the RPN arithmetic, stack manipulations, and Editor exercises in the first chapters. They are the cornerstones upon which you will build your Forth constructs.

With these tools you will be able to use Forth immediately, to sample the many techniques MMS has provided, and to review the programming itself. Filing of your MMSFORTH User Licensing Agreement will bring you additional information. Then you can subscribe to the MMSFORTH Newsletter, which will bring you a continuing stream. MMS has a variety of applications programs available and under development. We encourage use of these and marketing of other compatible software (provided a new MMSFORTH System Diskette is properly licensed to the same computer). Congratulations, you are now ready to "GO FORTH!"

A1.2 CUSTOMIZE YOUR MMSFORTH

In addition to the CUSTOMIZE Utility, this section describes some optional changes in source blocks, use of the Option Select Block, etc. Some combinations of these special features may be incompatible with each other or with your hardware or application. MMS support for such features is necessarily limited except on a consulting basis or through articles in the MMSFORTH Newsletter.

A1.2.1 The CUSTOMIZE Utility

The CUSTOMIZE Utility is more completely described in Appendix A4. Your initial MMSFORTH System Diskette may not be set to take best advantage of your particular hardware combination, because we must have it arrive at least minimally compatible with a maximum number of possible combinations. It is easy to customize a copy of your system to your own taste over a wide range of parameters. These resettable options may include memory size, low-block write-protect feature (PBLK), disk drives (number, drive motor start-up delay, individual track size, individual single/double density format settings on Model III and IBM/M.3 format settings on IBM PC, individual track stepping rates, 1 & 2 sided drives, etc.), an Auto-command sequence on boot, and more.

To CUSTOMIZE for booting a MMSFORTH System Disk on a Drive 0 with more than 40 tracks, temporarily use a 35- or 40-track drive AND one of the type you will be using. For example, add an 80-track drive above a 40-track Drive 0, put a formatted 80-track diskette into it (made with your 80-track DOS), and use our MMSFORTH master to do a BACKUP of itself from Drive 0 to the 80-track drive. Alternatively, add a 40-track drive above a 80-track Drive 0, and use your 80-track DOS special-copy utility (SUPERZAP's BACKUP, etc.) to bring the 35 or 40 tracks of MMSFORTH across to an 80-track formatted diskette. Then, with an 80-track as Drive 0, you can complete the job with another CUSTOMIZE.

Also use CUSTOMIZE to reprecompile additional utilities, extensions, application programs, cursor characteristics, lowercase on/off, and much more, so they are aboard upon boot. Get to know and use CUSTOMIZE early on.

A1.2.2 Some Block Modifications

Most MMSFORTH Utilities and Programs come set for a minimum of 32K RAM, but can be modified and for use in 16K systems. To adjust for 16K RAM, use a 32K or 48K Computer and EDIT out a left paren at the beginning of Line 1 (not line 0) of each affected utility and program. This normally FORGETs the Editor and often some more words which are not needed and free up enough space to permit the 16K application.

TRANSLATE won't fit 16K systems without major pruning, and some other programs might exhibit similar constraints.

Each time you adjust a block from the original, MMS recommends you place an asterisk (*) in the space immediately to the right of the date in Line 0. This provides simple yet prominent documentation in your future diskette indexes. Change your own block dates upon modification, but maintain the original dates on MMS blocks for future reference.

A1.2.3 **The Option Select Block**

As you gain expertise even more can be done by experimenting with the special features on the MMSFORTH Option Select Block, typically Block 15 on TRS-80 or Block 20 on IBM PC. MMS has shared these advanced features with the understanding that they may require unusual expertise and fine-tuning, which is available from MMS on a consulting basis. The TRS-80 Model I lowercase keyboard driver is the **only** such feature recommended for 16K RAM systems. Other features include variable number of block buffers, various printer-driver options for special printer capabilities, for using or avoiding the TRS-80 ROM printer-driver, and for system-level page formatting, an interrupt-driven keyboard with n-key type-ahead, and an alternate EXPECT which adds Editor-like features to the keyboard input line.

Once you understand CUSTOMIZE, use Block 15/20 as follows:

1. Using a backup copy of the MMSFORTH System Diskette without a write-protect tab, boot the system and 15 EDIT or 20 EDIT.

2. Add or remove leading parens and change the number of block buffers and value of DIRBLK as desired.

3. Press **Alternate-S** (on TRS-80, Shift-Clear-S).

4. Enter the code from Line 1 (now at the top of screen) and press Enter to reprecompile.

5. Complete the CUSTOMIZE operation when prompted to do so.

Briefly, the **Extended EXPECT** function permits editing of the keyboard input line. It is not the Editor, but acts like it. You will have the In-line Insert, Replace, Delete, Truncate, and right- and left-arrow functions. Try it to see which controls are aboard, and note that one new one is there, too: a **Alternate-R** will Retrieve the last-entered line for re-use or further editing!

V2.0's **interrupt-driven keyboard** option can be overly sensitive to some operations, so use it with care. It includes a type-ahead function which buffers up to 80 characters (adjustable) while your computer is otherwise engaged in disk I/O, processing a CATALOG listing, etc.

V2.1 incorporates an integral type-ahead buffer - on the IBM PC, it is 20 characters in size.

MMSFORTH's **variable block buffer** feature may be used to add one or many additional block buffers to your system - if you have enough available memory. This is a valuable feature for many special operations.

> TRS-80: If you have a printer which can print the TRS-80 graphics characters, modify the printer-driver block to send these. If you use the Epson printer-driver, do **not** also modify the screen-printer block similarly or you will offset the graphics codes by +32 **twice** instead of the intended once! You may opt to bypass the TRS-80 ROM printer-driver entirely - this permits faster multiple line feeds on some printers, while others work better with the ROM routine aboard.
>
> The serial printer-driver provides the above features in an alternate driver for serial (RS-232-C protocol) printers. Study its first block to see some of the more obvious adjustments it offers.

A1.2.4 System Constants

The System Constants Tables in Appendix A12 offer a vast quantity of additional system adjustments for special applications. The settings of many of these parameters are critical and will require careful adjustment by MMS or advanced users.

A1.3 DISK INDEXES

Your MMSFORTH diskette(s) will appear confusing when you first scan their contents, but you will want to find your way around the source blocks themselves, to load certain ones, etc. Many references in this manual give "typical" locations for certain blocks because these locations may be changed before new manuals are printed. For all these reasons, you need a handy index of your actual block locations. You can generate one live on screen with Forth's INDEX routine. The fancier TINDEX routine from MMSFORTH's CLOCK Extension has been used to generate a complete printed index of the System Diskette. It is delivered at the end of this Appendix, although you will probably choose to move the appropriate index to a new section, along with a TLISTS of your own. If you use ours instead of generating your own, be sure to check whether your disks have any more recent changes.

A1.4 DISK REORGANIZATION

Some original MMSFORTH disks are packed very full and will leave little space for your own programming. For your own regular use, you probably will want to set up one or more development or working diskettes which eliminate any source code blocks you will not be needing for the immediate job. Here is a professional way to do it.

Using these indexes and a sheet of paper, decide what blocks you want and where you want to put them. Then format your diskettes (at least a master and a backup). If you have more than one drive you can move ranges of blocks directly to the new disk using the COPIES Utility; if not, first backup all information onto the new disk and then use COPIES to shuffle it about. Use the BCOPY routine (press Break while in the COPIES Utility) to "smear" a formatted block across all those blocks you want to show as empty and ready for reuse. (For example, after using COPY to move an empy formatted block pattern to Block 80, use BCOPY to COPY onto Destination Block 81 for six blocks, starting from Source Block 80.)

If you have moved the directory and source blocks as a unit, the block relative load offsets will still be correct; otherwise calculate new ones and write them into the directory blocks, as well as commenting on their new absolute block numbers. Note the new PBLK value (the new lowest unprotected block number -- see A1.5.1), CUSTOMIZE your new system, and make the backup.

Finally, document your new system with a TINDEX.

A1.5 OTHER THOUGHTS

This concludes the list of necessary steps for beginning MMSFORTH and for adjusting the common options to your own preference. But there are other considerations useful to the beginner, and we will discuss some of them in this section.

A1.5.1 Block Protect

Disk MMSFORTH includes a useful word, PBLK, for software control of write-protection. By using it, your system software can remain write-protected while upper blocks of the same diskette are available for disk write operations. This feature effectively obsoletes the write-protect tab on most development diskettes in MMSFORTH, and makes practical single-disk applications packages which can update files while safeguarding system programming.

As delivered, your diskette typically protects all blocks lower than Block 86 (Model I), Block 118 (Model III) or Block 137 (IBM PC). This protects most of its programs but permits access to the data blocks for CHECKBOOK and LIFE. Verify this setting with PBLK ? . To change the protect limit to 32 during use, enter 32 PBLK ! . A like action is necessary before doing a normal write operation to any block lower than the current PBLK value. On reboot, PBLK is reset.

To permanently redefine the initial (precompiled) PBLK value on a disk system, use the CUSTOMIZE routine. (On the MMSFORTH Cassette System, first redefine PBLK and then save it with a FORTH-DUMP.)

A1.5.2 RAM Size

The MMSFORTH System Diskette is capable of effective operation on a single-drive TRS-80 with as little as 16K of random access memory. Most of the MMSFORTH demonstration programs do this; some rather large ones do so by the use of overlay techniques and clever programming, so look them over carefully and try these techniques yourself.

The main RAM-saving concept is to FORGET parts of the dictionary which occupy unproductive space. It may pay to move one or two existing word definitions into your source code, rather than to waste RAM space for the other words which share their blocks. Refer to Appendix 8, a CATALOG listing of MMSFORTH with full wordnames, to see which words will be still available after saying FORGET DIR, FORGET SCR, FORGET DIRBLK, etc. The listing also includes the RANDOM extension's wordnames, to demonstrate that extensions first FORGET DIR, then are loaded, and finally load DIR back on top. Additional extensions, whether called at the same time or later, behave in a similar manner.

TRS-80: Radio Shack has recommended that early Model I TRS-80's with 48K RAM **not** use the uppermost 6 bytes of RAM. MMS continues this recommendation.

A1.5.3 **Keyboard Debounce** (TRS-80 Model I, only)

Many early TRS-80 Model I's were delivered with poor tolerances between keyboard contacts. This problem was effectively masked in Level I BASIC systems but is accentuated by Level II's faster keyboard routines, which incorporate keyboard roll-over.

MMSFORTH's interrupt routine relieves this problem by slightly increasing the roll-over hesitation time. You also may wish to adjust and lubricate your keyboard to smooth its action without further loss of speed. (Do NOT attempt the following operation on the newer "ALPS" keyboards with rough-surfaced keytops, or on Model III's!)

Power down, then expose the keyboard switch contacts by carefully lifting off each keytop with a bent paper clip or a loop of string. Push down to close the contacts, while observing whether all four right-side contact fingers are closing simultaneously. If one leads, lags or is misaligned, carefully bend it back in line with a scriber, fine screwdriver, etc. Remove dust and corrosion if necessary. Then apply several drops of Archer Color TV Tuner Cleaner (Radio Shack Cat. No. 64-2320) or equivalent. Replace the keytop and press it quickly ten times to complete the operation.

An entire keyboard can be serviced in an hour or less. If you keep food, drink and cigarettes away from your keyboard and use a dust cover when it is not in use, it should perform excellently for at least a year between servicings.

A1.5.4 **Blinking Cursor & Auto-repeat Keyboard** (TRS-80 only)

It is possible to adjust the cursor character and its blink rate, as well as the auto-repeat characteristics. Memory locations for these parameters are found in the System Constant Tables of Appendix A12.

To change the cursor character, store an ASCII value into the variable CURSOR: 36 CURSOR !

The blinking cursor's on and off times may be changed independently, but the ratio of these values gives a readable cursor. If the cursor on-time is increased it will remain on longer, likewise if the cursor off-time is increased it will remain off longer. Therefore to slow the blink rate of the cursor increase both values proportionately.

A close approximation of a non-blinking cursor (unsuitable for editing) is available:

```
126 on-time C!  127 off-time C!
```

Or, for a standard TRS-80 Model I cursor:

```
14 EMIT  1 on-time C!  127 off-time C!
```

A1.5.5 **Lower-case Characters and Full-ASCII Keyboard**

MMSFORTH incorporates a lower-case display driver routine (optional on the TRS-80 Model I). Toggle it on or back off with **Shift-0** (Shift-Zero) on the TRS-80, with the CapsLock key on IBM PC. We recommend that you restrict your Forth words to upper case input, like the ones already provided.

MMSFORTH also incorporates a full-ASCII keyboard capability for outputting those characters unavailable from your keyboard's incomplete character set. This permits direct and complete control of certain printers, large-computer use via MODEM, etc. On TRS-80's use Clear for a Control key, and Shift-Uparrow for Escape. Note that MMSFORTH's floating-point arithmetic option uses a caret, (Control-" on TRS-80), as its symbol for exponentiation. A special-keys table is provided on the diskette, and a complete table in Appendix A11.

A1.5.6 **DRDSECS and DWTSECS**

MMSFORTH source code normally is reprecompiled with the CUSTOMIZE Utility. But precompiled code (machine code) may also be manipulated with DRDSECS and DWTSECS. These words read and write disk **sectors** to and from RAM. As you gain proficiency with MMSFORTH, you may find them useful to move sectors about, or just to analyze how CUSTOMIZE does it for you.

But, two warnings: DWTSECS overrides the PBLK protection, and can also overwrite important code with great ease when used by inexperienced hands. Experiment on expendable backups!

A1.5.7 **VIDEO ADJUSTMENT WITH HSYNC** (IBM PC only)

If your video display is not neatly centered horizontally, use n HSYNC to offst it to the best position. Try n=6 for a first try; that is enter 6 HSYNC (the default setting is n=0). If you want your System Disk to preset HSYNC your way, put the correct command in the AUTO command with CUSTOMIZE.

```
20 [20 :0] ( 04/30/82 IBM Option Select Block for recompiling MMSFORTH   )
21 [21 :0] ( 04/30/82 basic Compiler, 1 of 2 )
22 [22 :0] ( 04/30/82 basic Compiler, 2 of 2 )
23 [23 :0] ( 04/30/82 8088 ASSEMBLER,  1 of 15 )
24 [24 :0] ( 04/30/82 8088 ASSEMBLER,  2 of 15 )            (
25 [25 :0] ( 04/30/82 8088 ASSEMBLER,  3 of 15 )
26 [26 :0] ( 04/30/82 8088 ASSEMBLER,  4 of 15 )
27 [27 :0] ( 04/30/82 8088 ASSEMBLER,  5 of 15 )
28 [28 :0] ( 04/30/82 8088 ASSEMBLER,  6 of 15 )
29 [29 :0] ( 04/30/82 8088 ASSEMBLER,  7 of 15 )
30 [30 :0] ( 04/30/82 8088 ASSEMBLER,  8 of 15 )
31 [31 :0] ( 04/30/82 8088 ASSEMBLER,  9 of 15 )
32 [32 :0] ( 04/30/82 8088 ASSEMBLER, 10 of 15 )
33 [33 :0] ( 04/30/82 8088 ASSEMBLER, 11 of 15 )
34 [34 :0] ( 04/30/82 8088 ASSEMBLER, 12 of 15 )
35 [35 :0] ( 04/30/82 8088 ASSEMBLER, 13 of 15 )
36 [36 :0] ( 04/30/82 8088 ASSEMBLER, 14 of 15 )
37 [37 :0] ( 04/30/82 8088 ASSEMBLER, 15 of 15 )
38 [38 :0] ( 04/30 <CMOVE ENTER #IN Y/N SPACES PC! PAGE DIR CUR= UNLN-CUR)
39 [39 :0] ( 04/30/82 'S I I' J LOOP +LOOP DO LEAVE )
40 [40 :0] ( FILL ERASE BLANK NOT -TRAILING OUT/WORD CRT 2SWAP 2OVER D- )
41 [41 :0] ( 04/30/82 Input-Line Editor - alternate EXPECT, 1 of 3 )
42 [42 :0] ( 04/30/82 Input-Line Editor - alternate EXPECT, 2 of 3 )
43 [43 :0] ( 04/30/82 Input-Line Editor - alternate EXPECT, 3 of 3 )
44 [44 :0] ( 04/30/82 Parallel printer driver )
45 [45 :0] ( 04/30/82 Serial printer driver w/X-ON X-OFF protocol, 1 of 2)
46 [46 :0] ( 04/30/82 Serial printer driver w/X-ON X-OFF protocol, 2 of 2)
47 [47 :0] ( 04/30/82 PRINT PCRT empty-bufs FLUSH :R :0..3  DEPTH U.R U.)
48 [48 :0] ( 04/30/82 SET-WINDOW W/0 SET-COLOR SET-BORDER HSYNC IBM M.3  )
49 [49 :0] ( 04/30/82 PICK ROLL INP OUTP PC@ P@ P! 0C! 0C@ )
50 [50 :0] ( 04/30/82 B/W&COLOR   COLOR B/W TO-COLOR TO-B/W )
51 [51 :0] ( 04/30/82 * M* /MOD M/MOD  MOD /  M/  */ */MOD )
52 [52 :0] ( 04/30/82 DISP-B# BOX )
53 [53 :0] ( 04/30/82 SCR COPY DUMP ?TL TL LIST L BLIST PLIST PLISTS INDEX)
54 [54 :0] ( 04/30/82 .NAME CATALOG & 79-STD words )
55 [55 :0] ( 04/30/82 Screen EDITOR, 1 of 4 )          VOCABULARY EDITOR
56 [56 :0] ( 04/30/82 Screen EDITOR, 2 of 4 )             0 CONSTANT I/R
57 [57 :0] ( 04/30/82 Screen EDITOR, 3 of 4 )
58 [58 :0] ( 04/30/82 Screen EDITOR, 4 of 4 )      HEX
59 [59 :0] ( 04/30/82 IBM DIRectory Commands, 1 of 2 )
60 [60 :0] ( 04/30/82 IBM DIRectory Commands, 2 of 2 )
61 [61 :0] ( 04/30/82 EXTENSIONS Menu )      FORTH  PAGE
62 [62 :0] ( 04/30/82 DBL-PREC, 1 of 6 )
63 [63 :0] ( 04/30/82 DBL-PREC, 2 of 6 )
64 [64 :0] ( 04/30/82 DBL-PREC, 3 of 6 - stop load here if small RAM)
65 [65 :0] ( 04/30/82 DBL-PREC, 4 of 6: FORGET DU/MOD if req'd)
66 [66 :0] ( 04/30/82 DBL-PREC, 5 of 6: DU/MOD  D/MOD  D/ )
67 [67 :0] ( 04/30/82 DBL-PREC, 6 of 6: DU*  D*  D*/MOD  D*/ )
68 [68 :0] ( 04/30/82 ARRAYS )
69 [69 :0] ( 04/30/82 STRINGS, 1 of 3: ,$" $con $var $. $! $" IN$ $extrcts)
70 [70 :0] ( 04/30/82 STRINGS, 2 of 3: $+ $COMPARE INSTR )
71 [71 :0] ( 04/30/82 STRINGS, 3 of 3: $ to/from #  $arrays  $-TB )
72 [72 :0] ( 04/30/82 RANDOM numbers )
73 [73 :0] ( 04/30/82 GRAPHICS )  CODE ?BTM-SET  CX POP AX AX XOR 220 # CL
74 [74 :0] ( 04/30/82 SCREEN-PRINT for IBM )
```

```
75 [75 :0] ( 04/30/82 IBM-PC CASSETTE, 1 of 3 )
76 [76 :0] ( 04/30/82 IBM-PC CASSETTE, 2 of 3 )
77 [77 :0] ( 04/30/82 IBM-PC CASSETTE, 3 of 3 )
78 [78 :0] ( 04/30/82 IBM CLOCK, 1 of 2 )
79 [79 :0] ( 04/30/82     CLOCK, 2 of 2:  TIME DATE TLISTS TINDEX )
80 [80 :0] ( 04/30/82 TOOLKIT: .MEM  .S  TRY  RUN  2EDIT )
81 [81 :0] ( 04/30/82 Backup Drive 0 to 1 - add to TOOLKIT if you wish )
82 [82 :0] ( 04/30/82 BCOPY & B#IN routines - add to TOOLKIT if you wish )
83 [83 :0] ( 04/30/82 LONG-ADR, 1 of 3: S>L L2@ L2! L@ L! LC@ LC! )  HEX
84 [84 :0] ( 04/30/82 LONG-ADR, 2 of 3: LCMOVE L<CMOVE LFILL LWFILL )
85 [85 :0] ( 04/30/82 LONG-ADR, 3 of 3: LDUMP )
86 [86 :0] ( 04/30/82 UTILITIES Menu )
87 [87 :0] ( 06/23/82 IBM FORMAT, 1 of 4 )   : TASK ;
88 [88 :0] ( 04/30/82 IBM FORMAT, 2 of 4 )
89 [89 :0] ( 04/30/82 IBM FORMAT, 3 of 4 )
90 [90 :0] ( 04/30/82 IBM FORMAT, 4 of 4 )
91 [91 :0] ( 04/30/82 IBM BACKUP, 1 of 4 )                    : TASK ;
92 [92 :0] ( 04/30/82 IBM BACKUP, 2 of 4 )
93 [93 :0] ( 04/30/82     BACKUP, 3 of 4 - 4th is in IBM machine code )
94 [94 :0]
95 [95 :0] ( 04/30/82 This block is EMPTY )
96 [96 :0] ( 04/30/82 COPIES )
97 [97 :0] ( 04/30/82 SEARCH, 1 of 2; requires EEDIT )
98 [98 :0] ( 04/30/82 SEARCH, 2 of 2 )
99 [99 :0] ( 04/09/82 IBM CUSTOMIZE, 1 of 3 )  CREATE TASK 8 C, 4 C, 2 C,
100 [100 :0] ( 04/09/82 IBM CUSTOMIZE, 2 of 3 )
101 [101 :0] ( 04/18/82 IBM CUSTOMIZE, 3 of 3 )  BLOCK DROP
102 [102 :0] ( 04/30/82 PROGRAMS Menu )
103 [103 :0] ( 04/30/82 GUESS: number-guessing game )
104 [104 :0] ( 04/30/82 $SORT: String sorting demo )
105 [105 :0] ( 04/30/82 SORTS: Number sorting demo, 1 of 5 )  : TASK ;
106 [106 :0] ( 04/30/82 SORTS: Number sorting demo, 2 of 5 )
107 [107 :0] ( 04/30/82 SORTS: Number sorting demo, 3 of 5 )
108 [108 :0] ( 04/30/82 SORTS: Number sorting demo, 4 of 5 )
109 [109 :0] ( 04/30/82 SORTS: Number sorting demo, 5 of 5 )
110 [110 :0] ( 04/30/82 DOODLE, 1 of 3; also has data blocks )   : TASK ;
111 [111 :0] ( 04/30/82 DOODLE, 2 of 3; also has data blocks )
112 [112 :0] ( 04/30/82 DOODLE, 3 of 3; also has data blocks )
113 [113 :0] ( 04/30/82 LIFE, 1 of 2; also has data blocks )
114 [114 :0] ( 04/30/82 LIFE, 2 of 2; also has data blocks )
115 [115 :0] ( 04/30/82 Kaleidoscope, from BASIC program by D.Huntress )
116 [116 :0] ( 04/30/82 BREAKFORTH, 1 of 6, by Arnold Schaeffer )
117 [117 :0] ( 04/30/82 BREAKFORTH, 2 of 6 )
118 [118 :0] ( 04/30/82 BREAKFORTH, 3 of 6 )
119 [119 :0] ( 04/30/82 BREAKFORTH, 4 of 6 )
120 [120 :0] ( 04/30/82 BREAKFORTH, 5 of 6 )
121 [121 :0] ( 04/30/82 BREAKFORTH, 6 of 6 )
122 [122 :0] ( 04/30/82 SIMON, 1 of 3, by David Huntress )
123 [123 :0] ( 04/30/82 SIMON, 2 of 3 )
124 [124 :0] ( 04/30/82 SIMON, 3 of 3 )
```

```
125 [125 :0] ( 04/30/82 NOTEPAD Editor, 1 of 6 )  : TASK ;   HEX
126 [126 :0] ( 04/30/82 NOTEPAD Editor, 2 of 6 )
127 [127 :0] ( 04/30/82 NOTEPAD Editor, 3 of 6 )
128 [128 :0] ( 04/30/82 NOTEPAD Editor, 4 of 6 )
129 [129 :0] ( 04/30/82 NOTEPAD Editor, 5 of 6 )
130 [130 :0] : ES EDITOR CUR-POS 6 + C@  DUP ' W/EDIT 6 + C!  ' W/O 6 + C!
131 [131 :0] ( 04/30/82 CHECKBOOK, 1 of 6, plus 2 data blocks at end )
132 [132 :0] ( 04/30/82 CHECKBOOK, 2 of 6 )
133 [133 :0] ( 04/30/82 CHECKBOOK, 3 of 6 )
134 [134 :0] ( 06/10/82 CHECKBOOK, 4 of 6 )
135 [135 :0] ( 04/30/82 CHECKBOOK, 5 of 6 )
136 [136 :0] ( 04/30/82 CHECKBOOK, 6 of 6 )
137 [137 :0]
138 [138 :0]
139 [139 :0]
140 [140 :0]
141 [141 :0]
142 [142 :0]
143 [143 :0]              -|- MILLER MICROCOMPUTER SERVICES -|-
144 [144 :0]
145 [145 :0] PUTER ENVIRONMENT which simultaneously offers access in machine
146 [146 :0] custom programming tasks which are our bread and butter.  As
147 [147 :0] ( 06/28/82 PAINT, w/mods from TRS-80 version in Users Manual)
148 [148 :0] ( 04/30/82 SET-WINDOW options )
149 [149 :0] ( 04/30/82 Multiple-windows color display )
150 [150 :0] ( 04/30/82 DOTS demo )                        : TASK ;
151 [151 :0] ( 04/30/82 Demo routines )                        HEX
152 [152 :0] ( 04/30/82 Rough plot, 50X80 )                    DECIMAL
153 [153 :0]
154 [154 :0]
155 [155 :0]
156 [156 :0]
157 [157 :0]
158 [158 :0]
```

A3.0 MMSFORTH SYSTEM PROGRAMS

This section provides brief descriptions of each program in the PROGRAMS menu, seen by entering PROGRAMS upon booting the MMSFORTH System Diskette. (TRS-80 Model I: Swap in the Programs Diskette when prompted.) Some of these programs are designed as demonstrations of particular features, others are very usable programs. An examination of their source code blocks offers a wealth of Forth programming techniques.

A3.1 GUESS - NUMBER-GUESSING GAME

This classic program has you guess a number between one and a hundred. It demonstrates a use of the Random Numbers routines, and provides a good display of simple Forth programming techniques. You probably know how to write this program in another computer language, so this is an excellent starting example for analyzing our own Forth source code.

A3.2 $SORT - STRING-SORTING DEMO

This program is a demonstration of string sorting.

Enter a series of strings (names, animals, countries, etc.). End the series with a just carriage return (a "null entry"). The sort takes place **after** you press Enter, to give you a chance to set your stop-watch. Want to see it again?

A3.3 SORT - SORT ROUTINES COMPARED

This program allows you to compare various sort algorithms, and to **see** how they work in the computer. It's an unusually effective demonstration for computer classroom use, or to evaluate Forth's speed.

Select the type of sort from the SORT menu. Choose the number of items to sort. (128 or 160 is recommended for INSERTION and SELECTION sorts, 896 or 1840 for the other sorts.)

A series of random ASCII characters are generated directly into the video memory (and thus onto the screen), so the character generation and sorts take place before your very eyes.

A3.4 **LIFE** - GAME OF LIFE (AND DOODLE)

The Game of Life is a population dynamics simulation invented by John Conway, an Englishman, and first reported in the October 1970 Scientific American. The DOODLE sub-program allows you to make drawings on the screen.

Single cells, represented on the screen as graphics characters, reproduce, live or die governed by a simple set of rules based on how many direct neighbors abutt each cell. More than three neighbors and the cell dies of overcrowding, fewer than two and it dies of loneliness, otherwise it survives in the next generation. A cell will be born in any empty space which has exactly three neighbors.

When Life has loaded and its menu screen is on the video display, press L for the Load mode. You will be prompted for a block number at the bottom right of the video screen. Load the desired block. Typical block numbers are 75-77 on the Programs Disk for Model I; 120-123 for Model III, and 139 or 141 for IBM PC. You can use INDEX and EDIT to locate them on your disk.

You now can press G to Generate successive Life "generations", or you can press R to Reverse the white and black areas of the screen. Note that patterns with a lot of white cells will die out relatively fast. If you have started running Life and wish to stop it to do something different hold down the I (Interrupt) key until the blinking cursor returns in the upper right area of the screen. The Life program only scans the keyboard (to see if you are pressing the I key) when the full screen has been updated and before starting the next screen.

To enter your own pattern, note the location of the small block in the center of the screen (the drawing cursor) and the "compass rose", the small group of numbers in the upper right of the screen. The rose indicates direction: you press the number which represents the relative direction you wish to move the drawing cursor. The letter in the center of the rose indicates the mode of action: M for Move, D for Draw, or E for Erase. ROW and COLumn indicate the current position of the drawing cursor. The rose numbers correspond to the numbers on the computer's numeric keypad. Children (of all ages!) enjoy drawing pictures with this Doodle program.

If you wish to erase the screen press C to Clear it and W to make it White. Reverse can be performed repeatedly.

To save a particularly nice pattern or picture, press S for Save and you will be prompted (as with the Load option) for a block number. This block number is where you will write the contents of the video screen so be sure it is not one which has some important data on it. As delivered, the Model I MMSFORTH Programs Disk typically has 2 free blocks, 85 and 86, the Model III System Disk has 47 free blocks from 132-178, the IBM

PC System Disk has 12 free blocks from 147-158. (Although MMS may have furnished some extra information on these blocks, it can be accessed again from your original diskette when needed.)

Watch the MMSFORTH Newsletter for an occasional interesting LIFE pattern, or send us one!

A3.5 **ALIFE - LIFE WITH INNER LOOP REDONE IN ASSEMBLER** (TRS-80)

Demonstrates the extra speed which may be gained in any critical run-time Forth program by redoing inner, most-used loops into CODE routines using MMSFORTH's resident Assembler. Definitely run the longer-lived LIFE patterns in this mode.

> (ALIFE is not included on the IBM PC. LIFE's calculation time is already a small part of its display time, so speeding it up would have very little effect on the display rate.)

A3.6 **CHECKBOOK - CHECKBOOK BALANCING PROGRAM**

This program demonstrates the use of double-precision, simple scaling, and output formatting ("pictured" output). It has been selected for a detailed discussion in Chapter 8.

Always use Function 1 first for a demo session, or for repeat sessions with your own checkbook once it has been set up with beginning balances. Ending the program with Function 9 involves writing the new data to a ready diskette.

A3.7 BREAKFORTH - REAL-TIME VIDEO GAME, WITH SOUND

This now-famous and thoroughly enjoyable program supports the use of sound effects.

> TRS-80: To add this dimension to your fun, connect the computer's cassette output to a suitable speaker. One easy way to accomplish this is to attach an extension speaker to the EAR jack on your cassette recorder, and then to attach the computer/recorder cable to the recorder's AUX jack. Finally, fool the recorder into thinking it is recording a tape, by opening its tape compartment and holding in the tab at the back left while simultaneously depressing its PLAY and RECORD keys.

The object of BREAKFORTH is to remove as many "bricks" as possible from the mid-screen wall, by moving the bottom-line paddle sideways to return served balls - each brick disappears as the ball bounces off it. You control paddle motion with the TRS-80's Rightarrow and Leftarrow keys, or with the two right-most keys on the top row of the IBM PC's keyboard.

Your score grows as you remove bricks, with bricks from higher levels worth more. The ball speed increases as you break through to higher levels, and you will find that spin can be imparted to the ball by your paddle. If your skill and luck combine to remove all the bricks, you will get a surprise bonus and can continue to play the remaining balls.

Try the following BREAKFORTH game setting combinations: a speed of 3 and 5 balls to start, then a speed of 5 (7 on IBM) and 50 balls as your skills increase.

A similar, earlier version of BREAKFORTH is the subject of a major Forth article in the August 1980 special FORTH Issue of BYTE Magazine - don't miss reading its detailed analysis of this program! Arnold Schaeffer wrote this program on a MMSFORTH Cassette System as his first adventure into Forth, when he was a high-school junior. So get with it, Forth fans!

A3.8 NOTEPAD - ONE-PAGE LETTER WRITER PROGRAM

Last but not least, THE NOTEPAD is a useful gift to our users. At no additional cost, it's a complete text-editing program - not an elaborate one, but a fast, editor-like one in Forth source code ready to handle your letters, bills, etc. With a few adjustments, it will be ready for writing your Forth source blocks, as well, with some new wrinkles the full-screen Editor didn't even think of!

For standard use, consider THE NOTEPAD to have virtually all the same features as the regular Full-screen Editor. So master MMSFORTH's Editor operations, first. Then we need only describe the differences of THE NOTEPAD, in this section. Where the Editor only operates on a single 16-line **screen,** THE NOTEPAD addresses a range of blocks as a continuous **page** of linked screens which are temporarily stored in PAD independent of the Editor's block buffers. Its default mode links 4 screens into a page of 64 columns by 64 lines, a useful size for normal 8-1/2" x 11" paper. You can reset it at any time; to set it to 6 screens per page:

6 **#SCREENS** ! Easy? But leave it at 4, for now.

The sky's the limit on how you can use this new tool, but here are some starting ideas. First, load NOTEPAD from the menu and when it comes up enter a starting-block number followed by the same word, NOTEPAD, to read in some text from disk. We've included an important letter to you, starting typically on Model I Programs Diskette Block 81, Model III Block 128, and IBM Block 143. So read it in now, with 128 NOTEPAD . Notice the speed: it loads in about 1 second, and that you can immediately scan down four screens worth of lines with the downarrow! You can set up your printer and Print the entire contents with an **Alternate-P** (using the TRS-80 Shift and Clear keys for Alternate), or you can print any first portion of it by adding a Limit marker on the first column of the first non-printing line, with an **Alternate-L** while in the Line mode. Remove the limit marker when you're done using it, because you also can use it anywhere on a line to limit your Page mode Insert operations to a continuous "line" only up to that point. We find that for most operations, we like to be in both the Page and Insert modes at once. But all combinations are available and useful.

BRINGS UP NEXT BLOCK

Another new character is the Margin marker, invoked by an **Alternate-M.** While in the Page and Insert mode, put one in the first column of a paragraph you wish to move, then use the down-arrow only to proceed to the second column, first line of the following paragraph and put a second Margin marker there. Now put the cursor back on the first one and Delete the "line" with **Alternate-D.** Insert it wherever you prefer with an **Alternate-I,** and then Delete the Margin marker characters with a plain **Control-D** (Clear-D on the TRS-80) for each. Like it?

Your Save and FLUSH operations are handled together and immediately by Updating with an **Alternate-U.** This is different from the Editor's Update operation (which doesn't automatically FLUSH to disk), so be sure to have a formatted diskette mounted for the write operation. Check for write-protect tabs and improper PBLK settings, first. Update your working text often, for insurance! Once you've got your text written to the disk, you can Quit, with and **Alternate-Q.** Then you can use NOTEPAD to read another page of blocks from disk, or you can reread the same page by entering the short-hand word, **NP .**

When you are ready for more heady stuff, try setting #SCREENS to the length of the program you are working on, and edit it here, instead. You can express to the first or last word of the page with **Control-Uparrow** and **Control-Downarrow,** to the extreme columns of the present line with **Control-Leftarrow** and **Control-Rightarrow.** Wherever in the page you are, a **Alternate-Uparrow** (**Control-PgUp** on the IBM PC) or **Alternate-Downarrow** (**Control-PgDn** on the IBM PC) move you to the same position on the prior or following page, respectively (do an Update first, if you are saving the present text!). **Shift-Uparrow** and **Shift-Downarrow** are slightly modified from the Editor, in that they now move you to the next-found upper corner of a discrete block, ideal for tracking actual block locations when using THE NOTEPAD to edit a long "page" of Forth source code blocks.

Let's see your own MMSFORTH Newsletter items concerning programming tricks and practical applications for this exciting new MMSFORTH program!

A4.0 MMSFORTH SYSTEM UTILITIES

This chapter provides brief descriptions of each utility program on the MMSFORTH V2.0 or V2.1 System Diskette. These utilities are designed to load above the DIRectory for temporary use.

> 16K RAM: The FORMAT and BACKUP utilities need more memory than is available on a 16K RAM computer with a complete MMSFORTH system. Therefore, in order to run on 16K RAM systems, these two Utilities must first FORGET the Editor and redefine DIR to BOOT the system when done. Instructions for these 16K adjustments are given in the individual write-ups below.

A4.1 FORMAT

This utility is used to format blank diskettes. It is always necessary to run this program or a DOS-type format program before using any diskette for the first time. Since you may want a formatted diskette sometime when you are in the middle of something you don't want to interrupt, it is a good idea to format a few extra diskettes so you will have them handy.

A4.1.1 FORMAT - Operating Instructions

1. Bring up MMSFORTH System.
2. Type FORMAT, to bring in the FORMATting program.
3. The general flow of operation is:

Destination Drive (0..n) ?

where n is the highest number drive set in your system. (MMSFORTH does not check each drive to see if it is there; you must set the number of drives using the CUSTOMIZE Utility.)

Ready to format a n-track, single(double)-density disk
on Drive x (Y/N) ?

The system pauses for you to confirm that you have placed the correct disk in Drive n, without write-protect tab, etc.

Formatted Tracks: 0 1 2 ... n

The formatting takes place; each track number is displayed when it is written and successfully verified.

Format another disk (Y/N) ?

Allows you to start another. If the reply is Y it will go to "Ready to format..."

Put System Disk in Drive 0 (Enter)
Allows you to restore the MMSFORTH System Disk if you have removed it.

A4.1.2 **Advanced Users**

It is possible to use the FORMAT Utility to format any single track or any range of tracks. This can be particularly useful if you have a diskette which already contains data, and you wish to extend it from 35 to 40 tracks, or if you have an unrecoverable read error on a track on a diskette and want to re-format just that track.

In the last instance, it is important to remember that this utility works on **tracks,** not **blocks.** Here's how to calculate which tracks you want to format starting with a known block number:

> On TRS-80's, there are four standard 256-byte sectors per MMSFORTH Block, and ten 256-byte sectors per track on the standard TRS-80 Model I diskette or 18 sectors per track on the Model III. The first two sectors on the MMSFORTH diskette are used for the System Boot program and do not get counted in as part of a block; i.e., Block 0 starts at Track 0, Sector 2.
>
> A standard IBM PC diskette consists of 159 blocks per side: two 512-byte sectors per block, 8 512-byte sectors per track, 40 tracks per side. The first sector is a System Boot.
>
> Example: On what track is Block 20?
>
> Model I:
>
> $$\frac{(20 \text{ blocks} \times 4 \text{ secs/block} + 2 \text{ secs})}{10 \text{ secs/track}} = 8.2 \text{ tracks}$$
>
> = Track 8, Sectors 2-5
>
> Model III:
>
> $$\frac{(20 \text{ blocks} \times 4 \text{ secs/block} + 2 \text{ secs})}{18 \text{ secs/track}} = 4.56 \text{ tracks}$$
>
> = Track 4, Sectors 10-13
>
> IBM PC:
>
> $$\frac{(20 \text{ blocks} \times 2 \text{ secs/block} + 1 \text{ sec})}{8 \text{ secs/track}} = 5.25 \text{ tracks}$$
>
> = Track 5, Sectors 2-3

(**D?#** does this calculation automatically, but that won't help you to see what's going on!)

Prove it works, by reading 1024 bytes of data from this disk sector to the video display and comparing it to Block 20:

Model I: 15360 0 6 2 4 DRDSECS

Model III: PAD 0 3 8 4 DRDSECS
PAD 15360 1024 CMOVE
(In its normal mode, the Model III does not support direct screen I/O at double-density disk data rates.)

IBM PC: PAD 0 3 7 2 DRDSECS
PAGE PAD 1024 PUT-CHRS 20 0 PTC

To format a range of tracks or a single track, bring up the system and FORMAT utility as before. When the Destination Drive question appears on the screen, press the Break key to get MMSFORTH's ok prompt. Type:

drive# 1st-track# #tracks-to-be-formatted FORMAT-TRACKS

Press Enter and the system will format your tracks. For example, to change a 35-track diskette to a 40-track diskette on Drive 1, you will need to format Tracks 35 to 39. (In MMSFORTH, drive, track, and sector numbers always start at 0). First, be sure that the drive in that position is capable of writing to 40 tracks, and that you have customized your system to indicate that Drive 1 has 40-tracks. Then the command will be:

1 35 5 FORMAT-TRACKS

Voila!

A4.1.3 **Special Considerations**

TRS-80: The FORMAT Utility must be modified to allow running it in a 16K TRS-80 system. To do this, it FORGETs a major portion of MMSFORTH and redefines DIR to reboot the system. Borrow a similar computer with at least 32K RAM and edit out the beginning paren on Block 65, Line 1.

A4.2 **BACKUP**

This utility is used to backup one diskette to a formatted diskette. Unlike TRSDOS, BACKUP will **only** work to a diskette which is already formatted, and it will write over a diskette which contains data.

A4.2.1 **BACKUP - Operating Instructions**

1. Bring up MMSFORTH System.
2. Type BACKUP, to bring in the BACKUP program.
3. The general flow of operation is:

Source Drive (0..n) ?
where n is the highest number drive defined in your system. (MMSFORTH does not check each drive to see if it is there. You set the number of drives using the CUSTOMIZE Utility.)

Destination Drive (0..n) ?
(If source and destination are in the same drive, see Section A4.2.1.1.)

Ready to backup a n-track, single(double)-density disk
from Drive x to Drive y (Y/N) ?
The system pauses for you to verify that the correct disks are inserted, have no write-protect tabs on, etc.

Backed-up Tracks: 0 1 2 ... n
The backup takes place, an each track number is displayed when it is sucessfully transferred.

Backup another disk (Y/N) ?
Allows you to start another, or to FORGET TASK and return to the DIRectory. If the reply is Y it will go to "Ready to backup..."

Put System Disk in Drive 0 (Enter)
Allows you to restore the MMSFORTH System Disk if you have removed it.

A4.2.1.1 One-Drive BACKUP

If the source and destination disks are the same the procedure is a bit different. After answering the "Ready to backup ..." message you will get the message:

How many K RAM available (16..48) ?
Probable answer will be, 16, 32, or 48.

Ready to backup a n-track, single(double)-density disk
on Drive x (Y/N) ?
As with FORMAT this allows you to confirm that the right disk will be rewritten in the right manner.

Insert Source Disk (Enter)
At this point you should insert the disk which is to be copied into the appropriate drive.

Track: 00 01 02 xx
As many as will fit in the memory size you specified.

Insert Destination Disk (Enter)
Now put in the formatted diskette onto which you will be writing.

Track: 00 01 02 xx
Write out all those tracks which were read in.

Repeat the above cycle until all tracks have been copied through memory.

Backup another disk (Y/N) ?
Allows you to do another without having to start the program again. If you reply N ,

Put System Disk in Drive 0 (Enter)
allows you to restore the MMSFORTH System Disk if you have removed it.

A4.2.2 Errors

Possible error messages include reports that the disk drive number chosen by you is outside the range of drives that the system thinks it has, that the drive is not ready, or that you had a read or write error. A read or write error prints ?Read: or ?Write: followed by the number of the problem track.

A4.2.3 **Advanced Users**

It is possible to use the backup program to backup a single track or a range of tracks. This can be particularly useful if you have had to reformat a single track which contained important data, and you have a backup which has some obsolete data, but the particular track in question is still good, or good enough to use.

It is important to remember that this utility works on **tracks**, not **blocks**. (Use the COPIES Utility to copy blocks between single- and double-density diskettes on Model III, M.3 and IBM diskettes, on IBM PC, etc.) You can calculate which tracks you want to format, as described in Section A4.1.2.

To backup a range of tracks or a single track bring up the system as before. When the Source Drive question appears on the screen, press the Break key to get MMSFORTH's ok prompt. Place on the stack the following values:

source-drive# dest-drive# 1st-track# #tracks BACKUP-TRACKS

Press Enter and the system will backup your tracks. For example, to backup Tracks 13 and 14 from Drive 0 to Drive 1 the command will be:

0 1 13 2 BACKUP-TRACKS

Presto!

A4.2.4 **Special Considerations**

16K RAM: The BACKUP Utility must be modified to allow running it in a 16K TRS-80 system. To do this, it FORGETs a major portion of MMSFORTH and redefines DIR to reboot the system. Borrow a similar computer with at least 32K RAM, and edit out the beginning paren on Block 69, Line 1.

A4.3 COPIES

This utility is used to copy blocks from one part of a diskette to another or between diskettes. It will be invaluable for creating "slimmed down" system disks and reshuffling blocks for your own projects. Use it to copy between single-density (Model I) and double-density diskettes on TRS-80 Model III computers or between IBM and Model III formats on IBM PC computers.

A4.3.1 COPIES - Operating Instructions

1. Bring up MMSFORTH System.
2. Type COPIES, to load the Block Copying program.
3. If working between two different diskette formats, press Break, and configure drives as desired. (0 SDEN, 1 M.3, etc.) Then type COPIES once again.
4. The general flow of operation is:

First (lowest) source block# ?
Reply with block number to start copying from. This may be an absolute block number such as 135, or it may be a relative block number such as: 21 :1 .

Number of blocks to copy ?
Reply with the number of blocks you wish to copy.

First destination block# ?
Reply with the block number to start copying to.

Copy n blocks from Block x to Block y (Y/N) ?
The system pauses for you to **think** (as in IBM!) and to verify that the blocks and number of blocks are correct.

Count: 1 2 3 ... n
The copying takes place.

Copy other blocks (Y/N) ?
Allows you to do another without having to reload the program. If you reply N ,

Put System Disk in Drive 0 (Enter)
Allows you to restore the MMSFORTH System Disk if you have removed it.

A4.3.2 **Advanced Users**

BCOPY and <BCOPY are user words available within the COPIES utility. BCOPY copies the lowest block # first, while <BCOPY performs from the highest. (COPIES is "intelligent. It does whichever won't "smear"; i.e.,won't destroy data when the two block ranges overlap.) Just press Break to try them out. Some users prefer using these words directly, loading them into the TOOLKIT blocks for readiness (see an Index for block locations). Note that the :n words which permit relative block addressing may not be supported in the 16K RAM versions of some utilities.

These words expect three items on stack: from, to, and count. So, to use BCOPY type:

from-block# to-block# #blocks BCOPY

Use BCOPY or <BCOPY as you would use CMOVE and <CMOVE (see Glossary).

A4.3.3 **Error Messages**

Error messages concern block numbers which are off the disk or on a block which is software write-protected by PBLK, as well as read and write errors.

The only one likely to give any problems is a write error which will return you to MMSFORTH without completing the program. After the message Write: Disk Error: Block y ok you should enter FORGET TASK DIR to return to the menu or COPIES FORGET TASK DIR to reenter the program and prepare it to return to the Directory when completed.

A4.4 SEARCH

This utility is used to find a particular word in source blocks and optionally change it. It is useful if you wish to change all the references to a particular word for a new spelling, or to remove a routine which has become superfluous.

A4.4.1 SEARCH - Operating Instructions

1. Bring up MMSFORTH System.
2. Type SEARCH, to bring in the SEARCH program.
3. The general flow of operation is:

First block to search ?
Reply with the lowest block number you wish to search. The blocks must be contiguous, or you must make multiple passes through the program.

Number of blocks ?
Reply with the number of contiguous blocks through which you will be searching.

Search for ?
Reply with the word or phrase for which you are searching. It may be up to 31 characters, but may **not** include blanks.

Print the Matches (Y/N) ?
Allows you to have a list of the block and line numbers printed on a printer.

Edit the matches (Y/N) ?
If you chose N to the previous question, this question allows you to choose to edit the words as they are found. As soon as each match is found you are put into edit with the cursor on the particular word found and you can do any edit function. When all is edited to your satisfaction, press Alternate-U to mark the screen as **UPDATE**d, then Alternate-Q to go on to the next occurrence of the word. Be sure to type FLUSH when completed. This will force any screens still in the buffers out to disk.

A4.4.2 Special Considerations

16K RAM: The SEARCH Utility has provision to allow running it in a 16K TRS-80 system. To do this, you will have to change Block 75, Line 1 by removing the left-paren.

A4.5 **TRANSLATE UTILITY (FROM V1.9) - AND ALTERNATIVES** (TRS-80)

Because MMSFORTH Version 2.0 introduced the 79-STANDARD subset of Forth words, programs written in prior versions of MMSFORTH may require a few or even many corrections if they are to be moved up to this new system. MMS provides three ways to assist in this process: the TRANSLATE Utility, documentation to do it yourself, and our own services. MMS recommends the latter for our standard diskettes.

If Version 2.0 is the earliest MMSFORTH you have used, you can skip this section. The TRANSLATE Utility is used to translate TRS-80 MMSFORTH Version 1.9 source programs to Version 2.0. It should be run on each program **only once!** (It will translate **V2.0 code** into errors.)

A4.5.1 **THE TRANSLATE UTILITY** (TRS-80 only)

The TRANSLATE Utility will be the preferred tool for most of your own conversion needs. It is delivered as part of the MMSFORTH V2.0 System and will automatically rewrite many programs to that version, flagging nearly all of those few changes which may require your direct attention.

Slightly experienced Forth users should be able to use the TRANSLATE utility without additional documentation. Just call it up from your System Diskette's Utilities menu. If you have a printer, use it to record TRANSLATE's additional comments to you.

Like all new programs, experiment with this one before you expect total success. Be sure to work on a backup diskette so you can recover after an accident!

A4.5.2 **TRANSLATE - Operating Instructions** (TRS-80 only)

1. Bring up MMSFORTH System.
2. Before doing anything else, make a backup of the disk you intend to change. Do not TRANSLATE on the original! Otherwise, if you mistranslate the source code you might have no way to recover.
3. The general flow of operation is:

Put V1.9 disk(s), without write protect tab(s), into the drive(s) and press Enter

Pauses to allow you to insert the diskette(s) to be translated. Although one could do multiple diskettes in one pass, we recommend that you TRANSLATE one program at a time.

First block to translate ?

Reply with the absolute or relative block number at which to begin translation.

Number of blocks ?

Reply with the number of blocks to be translated in this pass.

Translate Blocks x thru y (Y/N) ?

Verifies the beginning and ending block numbers.

Output to printer (Y/N) ?

Allows you to choose to print the warning messages on a printer. UNLESS YOU HAVE NO PRINTER you should reply Y to this question. The warning messages are very important.

Translating to MMSFORTH V2.0

So you know that it is working.

Translate other blocks (Y/N) ?

Allows you to run another program through the TRANSLATE utility.

A4.5.3 **DOCUMENTATION FOR TRANSLATION** (TRS-80)

The next section, the rest of this USERS MANUAL, and the MMSFORTH Glossary provide information to enable you to edit changes into small programs and special projects, as well as to finish those items which the TRANSLATE utility brings to your attention.

Conversion of a V1.9 program to run on a V2.0 system involves consideration of three classes of words:

Class One: Words which behave exactly alike in both systems but have different names.

Class Two: Words which basically do the same function as before, but with a slightly different argument sequence.

Class Three: Words which existed in V1.9 but are not available in V2.0.

TRANSLATE handles Class One words by substituting them into the source text on the diskette. If the change will cause the source line to exceed 63 characters, the system will issue a warning message. If the new word is sufficiently long it might result in losing some overflowed text on that line, so be prepared to consult the original version when required.

A related point is that V1.9 compiled blocks as 16 lines of 64 characters, while V2.0 compiles blocks as a single 1024-character line. This means that values in the 64th column can become combined with the first character on the next line, as no space is inserted upon loading. This is good for some new uses (run-on quotes and comments), bad if it results in an unintended merged "word" such as ";:" . Similarly, be sure to **close** quotes and parenthetical comments if you do not want them to wrap around to additional lines! Closing " and) characters may be inserted in the 64th column.

TRANSLATE gives warning messages for the remaining two word classes.

The first two blocks of the TRANSLATE program may be edited to include other appropriate Class One or Classes Two and Three words, for your specific tasks.

CLASS ONE WORDS (replaced automatically)

V1.9 Word	**V2.0 Equivalent**
"	." (where used to open a quote)
',	[COMPILE]
(')	'
-DUP	?DUP
;S	EXIT
<BUILDS	CREATE
<R	>R
CLS	PAGE
DMINUS	DNEGATE
ECHO	EMIT
END	UNTIL
ERASE-CORE	EMPTY-BUFFERS
MINUS	NEGATE
MOVE	CMOVE
PEND	REPEAT
PERFORM	WHILE
R[	RP
WHILE	BEGIN

SOME SPECIAL CASES (not handled)

V1.9 Word	**V2.0 Action**
D!	2! uses reversed order for 4 bytes (no problem if new data).
D@	2@ as above.
DCONSTANT	2CONSTANT as above.
DVARIABLE	2VARIABLE as above, also is automatically initialized to 0. .
EDS	EDIT - no change.
MOD, /MOD, etc.	Remainder now has sign of divisor, not quotient.
Recursive definitions	Must use MYSELF.

SELDOM-USED V1.9 WORDS (not defined or different in V2.0)

#BL	#SB	$+LOOP	$;CODE	$BCASE
$LIT	$LOOP	&	(")	(L)
-'	32/16	?IN	ALF	ALFA
AM*	B*	C;:	CLEAR	CRBLK
CWBLK	D#	D#S	ERBLK	EWBLK
HLD	INTRP	LINE	M-	TEXT
TYPEIT				

CLASS TWO and THREE WORDS (warnings given)

Word	**Meaning of warning**
#TRACKS	Doesn't exist, see DISKDATA.
#DV	Doesn't exist, see DISKDATA.
-MOVE	Used ending (high) data area addresses as arguments. New word <CMOVE uses starting (low) addresses.
.	Printed 1 space before number in V1.9, prints 1 space after number in V2.0.
<#	Version 2.0 double-prec., etc. (see Glossary).
?	Printed 1 space before number in V1.9, prints 1 space after number in V2.0.
ACASE	Final " may erroneously be converted to ." by TRANSLATE.
BK1	Doesn't exist, see BUFFDATA.
BK2	Doesn't exist, see BUFFDATA.
CVARIABLE	V1.9 used initial value on TOS; V2.0 has no initial value.
EXECUTE	V1.9 used PFA (parameter field addr.), V2.0 uses CFA (code field addr.).
FILL	V1.9: chr start-adr count V2.0: start-adr count chr
H	V1.9 FORTH word for dictionary pointer and ASSEMBLER word for register. V2.0 FORTH is DP, ASSEMBLER is H .
HI#	Same definition in V2.0, but double-precision words now have high byte as TOS.
IMMEDIATE	V1.9 was an IMMEDIATE word, V2.0 is not.
LOAD	V1.9 Stacked a load request until next invocation of outer interpreter; V2.0 invokes outer interpreter when this word is executed.
LOADS	See LOAD; overridden by --> .
NCASE	See ACASE.
U*	V1.9: word * byte -> word V2.0: word * word -> double-word unsigned
U/	Not defined in V2.0
VARIABLE	V1.9 used initial value in TOS, V2.0 has no initial value.
WORD	V1.9 left nothing on stack, V2.0 leaves HERE on STACK.

A4.5.4 **MMS TRANSLATION SERVICES**

MMS can do it for you. Custom conversions must be charged by the hour, but MMS can easily rewrite your standard Model I MMSFORTH programs such as THE DATAHANDLER and the GAMES and UTILITIES diskettes for $10.00 each, plus shipping/handling, plus any increase in retail cost (none as of this writing). We recommend that you take advantage of this service in the case of these packages; it offers a low-cost opportunity to save work and to gain any features we may have added. Updated manuals, if desired, cost extra.

A4.6 **ALLCAPS** (TRS-80 only)

THIS UTILITY IS NECESSARY IF YOU DO NOT HAVE LOWER CASE CAPABILITIES ON YOUR MODEL I TRS-80. All the MMSFORTH programs and utilities can be used without lower case, because the display driver converts the characters as they are put out to the screen. HOWEVER, **BLOCKS CONTAINING LOWER-CASE CANNOT BE SUCCESSFULLY EDITED!!** The Editor does not go through the video display driver and uses the video memory for storage. Because the uppercase-only video display lacks one of its bits, any lowercase Forth source code which you don't convert to uppercase first will be converted to garbage, unreadable when on the screen and converted to bad data if saved back to disk.

A4.6.1 **ALLCAPS OPERATING INSTRUCTIONS** (TRS-80)

1. Bring up MMSFORTH System.
2. Make a backup of the disk you intend to change.
3. Type ALLCAPS, to bring in the Capitalizing program.
4. The general flow of operation is:

PUT DISK TO BE CAPITALIZED INTO DRIVE 0
FIRST BLOCK TO BE CAPITALIZED ?
Reply with the block number of the first block to be capitalized.

NUMBER OF BLOCKS ?
Reply with the number of blocks to be translated. DO NOT RUN THE ALLCAPS UTILITY ON MACHINE LANGUAGE BLOCKS, or they will be incorrectly modified to garbage.

READY TO CAPITALIZE BLOCKS x THRU y (Y/N) ?
Allows you to verify that the right blocks are chosen.

CAPITALIZING BLOCK: 15 16 17 ... nn
The capitalizing takes place.

CAPITALIZE OTHER BLOCKS (Y/N) ?
Allows you to start another pass.

A4.7 CUSTOMIZE

This utility is used to reset the system parameters including number and type of disk drives, memory size, software block protect, auto-command(s) on boot, directory block location, and printer margin. It can be used to permanently incorporate new system or user routines into the portion which comes up on Boot.

A4.7.1 CUSTOMIZE - Operating Instructions

1. Bring up your MMSFORTH System.

2. Enter FORGET DIR (unless you want the final disk to use the old DIRectory).

3. Load any system routines and user programs which you want to include in the booted (precompiled) portion of the system.

4. If you typed FORGET DIR before, type DIR now so CUSTOMIZE is defined.

5. Type CUSTOMIZE, to bring in the CUSTOMIZEing program (it does a FORGET DIR).

6. CUSTOMIZE itself goes like this:

First available Block#, N, will be x
where x is determined by the dictionary size.

Directory Block# ?
The block number which will be initialized into DIRBLK should be entered here. Typically, this value is 40 on Model I and Model III systems, 59 on IBM PC. You can confirm this by displaying the value of the DIRBLK constant.

Lowest unprotected Block# ?
Sets the software write-protect feature. Any track numbers lower than the number entered here cannot be written to without first lowering the contents of PBLK, the variable where this number is stored. Typically, the Model I system comes with PBLK set to 86, (leaving just Block 86 empty on a 35-track standard disk!), and the Model III system to 118, and the IBM PC to 137. You can confirm this value with PBLK ? .

> Note: The values in these sample lines may vary depending on your computer.

Memory size (-32768=16K, -16384=32K, -6=48K) ?
Sets the memory size for the system. Caution: setting this number larger than available RAM will make your disk inoperable! If you wish to load a printer or other driver in high memory you would reduce the address

entered by the size of that program.

Printer left margin (0 is std.) ?
Allows you to set a software margin on the printer. This is especially nice if you have a printer with unmovable tractors. It allows you to indent the left margin for centering Forth blocks, or to punch for looseleaf binding.

Number of block buffers is 2
If you had reset the number of buffers in Block 15, Line 4 (IBM PC Block 20, line 4) this would give you a choice of 2..n buffers to be allocated. To change the number of allowable buffers, change Block 15, Line 4 from 2 BUFFDATA to n BUFFDATA where n is a hexadecimal (base 16) number representing the maximum number of buffers you wish to allow for. If you change Block 15, you must first reprecompile the system according to the instructions on Line 1 of Block 15, before using CUSTOMIZE to make the change permanent.

Disk start-up speed (1...255, 48 is std.) ?
Allows you to change the delay time between turning on the disk and actually reading or writing. The system default is a carefully calculated trade-off between adequate delay to allow writing, and minimized delay to keep down operating time. A few disk drives may give read/write problems, especially if run on the 50-Hertz current used in many European countries. These drives should run reliably if this delay number is increased to 100 or 150.

# of disk drives (1..4) ?
Allows you to set the number of disk drives. This will allow you to access as many disk drives as you have set here. If you have set the number of disk drives to 2 and you really only have one drive, you will get a read error upon trying to access the non-existent drive, but if you have set the number of drives to 1, you will get a Block out-of-range message which will not mess up your buffer contents. If you have more physical drives than are specified here you will not be able to access the extra drives.

Drive 0: Single or Double density (S/D) ? (Model III only)
Specify initial density, normally D.

Or, Drive 0: IBM or M.3 (I/M) ? (IBM PC only)
Specify initial format, normally I. Permits use of TRS-80 Model III formatted diskettes (179 blocks/side!) except in Drive 0 during boot.

Number of tracks (40 is std., format must match!) ?
Set the number of tracks which your drive will be able to access. Most drives can access at least 40 tracks, but older Radio Shack Model I drives access only 35. You can use the FORMAT-TRACKS portion of the FORMAT utility to add higher tracks on a previously 35-track diskette.
TRS-80: Repeats last two for each drive.

Disk speed (0=6, 1=12, 2=20, 3=40 msecs; 3 is std.) ?
You can set the track access speed higher on some drives than on others. Model I TRSDOS is set for the slowest speed because the old Radio Shack drives could only go that fast. You will get read and/or write errors if you try to run at a speed higher than your drives can handle. Some typical Model I drive speeds are: Older Radio Shack (Shugart) = 3, BASF, Percom and TEAC = 2, MPI & Tandon = 1. IBM PC drives, Model III delivered Radio Shack drives, MPI and Tandon drives and some others can run at 0. If in doubt, experiment carefully.

IBM PC: Repeats last three for each drive.

Auto-command:
Allows you to set up a command or series of commands which will be executed on Boot after the copyright message is displayed. The Auto-command string may be a maximum 80 characters (but it could specify 150 LOAD, for example, to do more). For a standard system, reply DIR . This feature is similar to the TRSDOS AUTO command, or the PCDOS AUTOEXEC.BAT File. To temporarily suppress this feature on the TRS-80, hold down the Break key while booting (if you get an error message later, enter EMPTY-BUFFERS before proceeding). On the IBM PC hold down Control-NumLock, instead.

Place disk to be CUSTOMIZED in Drive 0 (Enter)
Place disk in drive; the CUSTOMIZE utility will write the boot and precompiled system onto it.

A4.7.2 **Temporary CUSTOMIZE**

It is possible to run the CUSTOMIZE program up to the Auto-command question, then press Break and have your system retain the new settings in memory. They will not be on the disk, so the old values will return when you reboot. If you need to temporarily set something different, this is one way to do it.

A5.0 MMSFORTH SYSTEM EXTENSIONS

Certain extension wordsets for MMSFORTH are available as added-cost packages. For example, the MMSFORTH Utilities Diskette includes a Floating-point Math wordset, a full Z80 Assembler wordset (TRS-80 only), and a Cross-referencer Utility. In this Appendix, however, extensions means special included wordsets for optional addition to MMSFORTH's menu on the Systems Diskette.

You can quickly find the Forth source code location of each set of extension blocks by consulting an INDEX, or by looking them up on the directory blocks:

```
DIRBLK EDIT
```

and one block higher. Examine the contents of these extension blocks with your Editor, learn their operation from your Glossary, and **use** them to gain familiarity.

When one or more Extensions are called from the MMSFORTH System Diskette DIRectory, the Extensions are loaded after a FORGET DIR and the DIRectory words are then reloaded (see Appendix 8).

A5.1 DOUBLE-PRECISION NUMBERS

Users new to Forth are often surprised by its preference for integer mathematics. There are good reasons for preferring single-precision integer arithmetic most of the time, double-precision when needed, and floating-point as a rare exception.

Single-precision integer math is sufficient for most purposes and where it fits, it fits smallest, fastest, easiest and without floating-point's inherent round-off error. For most business accounting applications and the like, single-precision's values from 0 to 65535 or from -32768 to 32767 won't hack it, but double-precision will count over an integer range of more than +/- two billion, or will count the pennies across a range of better than +/- $21,000,000.00 . (If your assets exceed this amount, you can afford to hire MMS to add triple-precision routines for your use!) Although that decimal point isn't printed out by the double-precision operation, it is easily added to the output with an appropriate word. (See MMSFORTH's CHECKBOOK balancing program for an example.) Again, double-precision integer math outperforms floating-point in compactness, speed and lack of round-off error.

Unlike most versions of Forth, MMSFORTH does offer a floating-point package as an option. This option leans heavily on your computer's BASIC ROM to save space and speed (it's in fast machine code and it's already aboard, so no extra RAM). For numbers which vary over a wide dynamic range, think **scaling** for advanced users (there is an example in the

CHECKBOOK program) or floating-point for beginners. For more usual applications, MMS strongly recommends you get to meet your double-precision instruction set.

MMSFORTH's double-precision word set is loaded by entering DBL-PREC . This loads six blocks of source code. If RAM space is limited, consider loading the first three blocks, which will provide all functions except D/ , D/MOD , D* , D*/ , and D*/MOD .

A DOUBLE-PRECISION NUMBER IS ENTERED IN MMSFORTH BY INCLUDING A DECIMAL POINT IN THE NUMERIC INPUT. Thus, 5 places a single single-precision element on the stack, while 5. places a pair of elements on the stack. In keeping with 79-STANDARD, the double-precision representation has the top-of-stack equal to the most significant 16 bits of the number and the second-on-stack is the least significant 16 bits. Actually, all numbers entered in MMSFORTH are remembered as double-precision quantities. The value of the byte variable **#PT** indicates if one or two values have been pushed on the stack. If #PT = 0 then one value has been pushed on the stack because no decimal point was included in the input; the variable **HI#** contains the high order bits of this number and can be used to check for overflow or to create a double-precision entry by HI# @ SWAP . If #PT <> 0 then two values have been pushed on the stack and #PT indicates the position of the decimal point in the input. #PT=1 if the decimal point was the last character input, =2 if next to last, etc. Thus, 5. has #PT=1 ; 5.7 has #PT=2 .

Double-precision constants are defined by putting an initial double-precision value on the stack and using the word 2CONSTANT to enter it into the dictionary:

```
5 2CONSTANT XX
```

Double-precision variables are automatically initialized with a value of 0.:

```
2VARIABLE FIVE   5. FIVE 2!
```

Note: Although MMSFORTH normally initializes variables (CVARIABLE , VARIABLE , 2VARIABLE , etc.) with a zero, 79-STANDARD does not require this; it is good general practice to store in a 0 or other value before other use.

A5.1.1 TABLE OF DOUBLE-PRECISION ARITHMETIC OPERATIONS

Stack inputs and outputs are shown with top of stack on right.

OPERAND KEYS:

d,d1,...	double prec. #'s	ub1,ub2	unsigned byte #'s.
n,n1,...	single prec. #'s	un1,un2	unsigned single prec. #'s.
f,f1,...	flag: true=1, false=0.	ud1,ud2	unsigned double prec. #'s.
addr	address	ut1,ut2	unsigned triple prec. #'s.

Word	Stack	Action
M+	(d1 n1 -> dsum)	add single to double.
D+	(d1 d2 -> dsum)	add double to double.
M-	(d1 n1 -> ddif)	subtract single from double.
D-	(d1 d2 -> ddif)	subtract (d1-d2).
M*	(n1 n2 -> dprod)	multiply singles ans. double.
D*	(d1 d2 -> dprod)	multiply doubles ans. double.
M/	(d1 n1 -> nquot)	(d1/n1) single result.
D/	(d1 d2 -> d3)	(d1/d2) double result.
M/MOD	(d1 n1 -> nq nr)	(d1/n1) single quot + remainder.
D/MOD	(d1 d2 -> dq dr)	(d1/d2) double quot + remainder.
M*/	(d1 n1 n2 -> d2)	(d2=d1*n1/n2)
D*/	(d1 d2 d3 -> d4)	(d4=d1*d2/d3)
D*/MOD	(d1 d2 d3 -> d4 d5)	as above, but d4 is remainder.
DU/MOD	(uq ud1 -> ud2 ud3)	(uq/ud1) d quot. d remainder.
DU*	(ud1 ud2 -> uq)	(uq = ud1*ud2).
D*S	(ud un -> ut)	(ut=ud*un).
T/S	(ut un -> ud)	(ud=ut/un).
DABS	(d1 -> d2)	d2= abs(d1).
DNEGATE	(d1 -> d2)	d2=-d1.
D0=	(d -> f)	f=1 if d=0. else f=0.
D<	(d1 d2 -> f)	f=1 if d1 < d2.
DU<	(ud1 ud2 -> f)	f=1 if ud1 < ud2.
D=	(d1 d2 -> f)	f=1 if d1 = d2.
DMIN	(d1 d2 -> d3)	d3= minimum of d1 or d2.
DMAX	(d1 d2 -> d3)	d3= maximum of d1 or d2.
2@	(addr -> d1)	fetch double contents.
2!	(d addr ->)	store double quantity.
2DUP	(d -> d d)	duplicate dquantity on stack.
2DROP	(d ->)	throw away double quantity.
2SWAP	(d1 d2 -> d2 d1)	reverse top 2 double items.
2OVER	(d1 d2 -> d1 d2 d1)	make copy of 2nd double on stack.
2ROT	(d1 d2 d3 -> d2 d3 d1)	rotates 3rd number to TOS.
2CONSTANT x	(d ->)	create double constant x w/value d.
2VARIABLE y	(->)	create double variable y.
D.	(d ->)	print double number.
D.R	(d ub ->)	print d right justified in fld ub.
D#IN	(-> d)	input double number from keyboard; if no "." convert to double.

A5.2 **ARRAYS**

The ARRAYS words are generally described in Section 4.4, and practical examples of their use are scattered throughout the MMSFORTH source code, the sample programs in Chapters 7 and 8, etc. Try using the SEARCH Utility to locate good examples for study.

A5.3 **STRINGS**

MMSFORTH implements strings in a manner generally compatible with Radio Shack (Microsoft) Extended BASIC. Read Section 5.2, then try your own experiments and look over the Programs Menu for good examples of use.

A5.4 **RANDOM**

Use RANDOMIZE to start your "random" sequence at an unpredictable point, then select values from 1 to n with n RND .

A5.5 **GRAPHICS**

MMSFORTH offers several types of graphics, including the usual use of alphanumerics for pictures and graphs, the EMITting of graphics ASCII codes such as 191 on TRS-80 and 219 on IBM PC for complete white (see the PAINT example in Chapter 7), etc. The GRAPHICS Extension adds the equivalent of the TRS-80 BASIC graphics words SET, RESET, and POINT: ESET, ECLEAR, and E?.

TRS-80: A double-width graphics set, DSET, DCLEAR and D? , is also supported. Although the latter set has only half the horizontal resolution, its uniform horizontal and vertical resolution scales simplify many displays.

IBM PC: MMSFORTH's low-resolution (ESET) graphics wordset can be mixed with text. It offers improvements over IBM's own graphics, including 16 instead of 4 colors supported at once (if your color display "sees" the intensity bit), a compatible display on the monochrome display, and the ability to transport existing MMSFORTH games such as Breakforth, etc.

A5.6 **SCREEN-PRINT**

Load this extension for direct dumping of the screen display to your printer. It's ideal for documenting your early experiments or your problems for later debugging. If you have a printer which can reproduce the TRS-80 graphics characters (Okidata, Epson, etc.), edit the printer-driver and/or this extension block to take advantage of your good luck. Try it on the ALIFE program (LIFE on IBM PC). If it won't load first, begin by loading ALIFE, then interrupt it with a Break, 55 LOAD (typical), and restart with ALIFE . This will be necessary anytime a newly-loaded program FORGETs beneath the routine you have loaded.

A5.7 **CASSETTE**

The MMSFORTH System Diskette is equipped with this extension for communication with MMSFORTH Cassette Systems, and for using tape as an alternate data storage medium.

> Model III Systems: May run at 500 baud (for Model I tape format) or 1500 baud, selected by **H/L**; then respond **H** for high speed, or **L** for low.
>
> IBM PC: Implements the standard IBM PC physical tape format, normally only for use as an alternate data storage medium.

DISK-TAPE will move a range of blocks to tape, while **TAPE-DISK** will move them in the other direction. RBLK and WBLK may be toggled to the **TAPE** mode or back to **DISK**, by invoking either of these words.

A5.8 CLOCK

This extension offers time and date capabilities, plus several words which join them into report lines for a titled INDEX (**TINDEX**) and a titled LISTS (**TLISTS**). You must remember to initialize with **SET-TIME** and **SET-DATE** before using them.

A5.9 TOOLKIT

Some very useful additional routines are kept in this "catch-all" extension. Bring it in for debugging, etc. Change its words according to your preference, or add additional blocks as required. (Two following blocks are located there for this purpose.)

Try our words **.S** and **TRY** to learn how the stack operates, and use them later for debugging your advanced stack-manipulating words.

Use **2EDIT** and **2EDITS** to compare two versions of the same block or two versions each of a series of blocks, respectively. These routines are ideal for locating small changes in two versions of a program, etc. They set up a matched pair of blocks for use with Alternate-E, the Editor's Exchange function. Just flicker the two blocks and any changes will show dramatically. Then edit and Save the screen, or Quit and proceed to the next set.

A5.10 **LONG-ADDRESS WORDS** (IBM PC ONLY)

MMSFORTH, like all 79-STANDARD versions, handles its memory addresses as single-precision (16-bit) numbers. Thus, the highest address it can directly identify is 65535, or "64K".

64K is a large enough span to accomodate all possible memory addresses on the TRS-80, and enough addresses for any practical Forth program. However, the IBM PC permits up to a Megabyte of RAM. This additional capacity is not needed for program space, but it can prove of value for the storage and manipulation of large files, etc. So MMSFORTH V2.1 provides another extension, **LONG-ADR**, to permit double-precision (32-bit) memory operations.

Most of the LONG-ADR wordset has an obvious and direct relationship to its single-precision equivalent words: **LC@ , L@ , L2@ , LC! , L! , L2! , LCMOVE** and **L<CMOVE . LFILL** and **LWFILL** are both equivalent to FILL except that LFILL takes the long-address (32-bit) source, word (16-bit) count, and 8-bit character to be filled, while LWFILL takes the long-address (32-bit) source, word (16-bit) count, and 16-bit word to be filled. **LDUMP** is the long-address equivalent to DUMP , taking a double-precision (32-bit) address and a word (16-bit) count.

S>L is a final word in the LONG-ADR Extension wordset. It converts "short-to-long"; that is, a short (16-bit) address to a long (32-bit) address.

A5.11 **B/W&COLOR** (IBM PC ONLY)

This optional extension block is only for the special case where a Color Graphics Adapter and a Monochrome Adapter are installed together in one IBM PC computer. In this case, compiling this block permits intelligent manipulation of both boards and their respect video displays by the Forth program or from the keyboard.

Booting the system with this option compiled will default to the monitor-type settings on Switch 1 of the IBM PC's System Board. The words **COLOR** and **B/W** reinitialize and switch over to their respective displays.

You may wish to shift video output between the two displays, without reinitializing the "new" screen's setting; i.e., in displays where the prior settings, whatever they were, should be continued. Two other words, **TO-COLOR** and **TO-B/W**, are provided for this purpose.

A5.12 **MORE IBM PC WORDS: BOX, SET-COLOR, SET-WINDOW**

The following words are not properly extension wordsets, as they are compiled within the IBM PC's standard MMSFORTH V2.1 wordset. Because they are new to MMSFORTH, for IBM PC only and deserve special mention as such, we will treat them here as if they are another special wordset.

BOX is used for the special window borders in the Full-screen Editor (both the main one and the view of the one-line buffer at PAD), in NOTEPAD, etc. Its parameters are described in the comment adjacent to its definition on Block 52. For an interesting graphics use of BOX, try the following:

```
: BOXES   0 DO  10 4 10 I I 3 * BOX  LOOP ;
PAGE  18 BOX
```

SET-COLOR defines the present video display attributes - color, blinking, color reversal, underline in monochrome, etc. It takes the background and then the foreground color code from stack. Monochrome examples: 0 7 SET-COLOR for standard, 7 0 SET-COLOR for reverse video, 0 15 SET-COLOR for high-intensity, 0 1 SET-COLOR for underline, 0 9 SET-COLOR for high-intensity **and** underline, etc. Color examples: 0 7 SET-COLOR for standard, 7 0 SET-COLOR for reverse video, 8 7 SET-COLOR for blinking, 0 15 SET-COLOR for bright white, 4 14 SET-COLOR for more color, etc. See the hardware documentation (or the BASIC word, COLOR) for more information on attributes, or analyze our examples on the System Diskette.

On a color display, **SET-BORDER** takes a single color code for the border setting, in a complementary manner to SET-COLOR.

COLOR? tests which display board is active to see if color is available. If so, it returns a 1; if not, a 0. See the Kaleidoscope demonstration program, KSCOPE, for a clever use of COLOR? in conjunction with a random NEW-COLOR word.

SET-WINDOW defines windows such as W/0, the default window. The series of windows defined in the demonstration program on Block 148 looks spectacular in color, and shows some of the flexibility of this screen-allocation technique. Note that the second parameter defined is the combined background-foreground color codes for the window; in HEX, the two values are the two digits.

A5.13 **LIST OF NEW IBM PC WORDS**

The following are new MMSFORTH Version 2.1 words introduced for the IBM Personal Computer. Their definitions may be found in the MMSFORTH Glossary, Appendix A9.

ENCODE	D?#	DISP-B#	BOX	PINIT
P-IT	P&C	OUT/WORD	?CLR	CRESET
HSYNC	SCAN-CODE	GET-CHR	#CHRS	
MOVE-CHRS-TO-SCRN		MOVE-CHRS-FROM-SCRN		CUR-POS
GTC	C=	UNLN-CUR	FULL-CUR	NO-CUR
PUT-CHRS	GET-CHRS	COLOR?	COLOR	B/W
TO-COLOR	TO-B/W	>B:D	B:D>	SET-WINDOW
SET-COLOR	SET-MODE	SET-BORDER	IBM	M.3
PC@	PC!	P@	P!	
0C!	0C@			

A6.0 TROUBLE SHOOTING IN MMSFORTH

Well, yes, occasionally something might go wrong! The most common occurrences are error messages for improper entries or hardware problems. The most frustrating are the ones which lock up the system. All will be overcome as you learn good trouble-shooting techniques and the proper ways to use your new programming environment. In fact, for the experienced Forth programmer, debugging can be great fun!

MMSFORTH includes some checks against the most common Forth errors. For example, accidentally hitting a non-printable character such as backarrow while defining a wordname will not show on most other systems, but makes using the word later quite impossible; so MMSFORTH rejects non-printing keys while you are keying in wordnames, substituting an underline character as a warning, instead. And our Forth routines and programs usually include last-minute "Are you sure?"-type queries to the user just before potentially painful mistakes.

Because each protection feature costs space, development time, and/or user flexibility, too many can become a problem. And of course, there is no way to keep you from occasionally making a mistake. But planning ahead will help. Be especially sure that you return from HEX or another number base before doing important write operations, or they will end up on the wrong block and that block's prior contents will disappear! Saying DECIMAL an extra time costs little, compared to a loss of data. The best solution by far is to have at least one backup of any important work, and to keep it intact until determining that the new diskette is also in good shape. IBM says, "THINK!"; MMS says, "**BACKUP!**"

A6.1 ERROR MESSAGES

MMSFORTH has simple and effective diagnostic messages. They tell you what the problem is, and if they occur while compiling from blocks they also report the block, line and column numbers at which the offending word was found. If you wish, you then can directly "error-edit" that block by entering "**EEDIT**". Error messages are **not** terminated with an ok .

A6.1.1 Compile Error Messages

xxx ? This most common FORTH error means the word xxx is not in the dictionary and cannot be converted to a valid number using the current BASE. Not in the dictionary does not mean the word is unknown to the Forth system, but that it cannot be found in the particular **vocabularies** it searched.

> A not-so-simple explanation of vocabularies: FORTH is the trunk vocabulary and vocabularies defined in FORTH form branches from this trunk; for example, EDITOR and ASSEMBLER. We consider FORTH the father branch; EDITOR and ASSEMBLER are son branches of FORTH and sibling branches to each other. Vocabularies defined in branch vocabularies form sub-branches. A search proceeds from the "CONTEXT" branch and continues to the trunk through all father, grand-father, great-grand-father branches but not its siblings or siblings of its father, grand-father, etc.

A vocabulary is made the "CONTEXT" vocabulary by using its name. Words are defined into a vocabulary by using its name followed by "DEFINITIONS"; e.g., "EDITOR DEFINITIONS". After this statement all new words are entered into the EDITOR vocabulary, until a new DEFINITIONS clause is invoked. You can confirm which words are in which vocabulary: in theory by consulting the Glossary, in practice by entering FORTH CATALOG or EDITOR CATALOG , etc. Remember to reenter FORTH when you are done.

; ? This compile (semicolon) error is caused by unbalanced conditional or loop structures. Forth is saying, "You're not **really** ready to compile that definition, are you?" Look for a missing ; , THEN , UNTIL , or similar conditional imbalance.

?Dup-name: xxx is a warning issued when compiling a word whose name has already been used and is found in the current search path. Duplicate names are allowed and are quite desirable in some instances; this warning is given because you may be unable to access the previous definition when this new word is also in the dictionary. To inhibit ?Dup-name: messages from your final application programs:

```
0  21 MMS 25 +  C!
```

Stack empty is an error message given when the outer interpreter recognizes that the stack pointer is beyond its lower limit. Note that it only recognizes the problem when the outer interpreter sees it; thus, be careful not to underflow the stack with a bad routine such as:

```
: TROUBLE   BEGIN  .  0 UNTIL ;
```

See Section 6.2.1 for some related no-no's.

Dictionary full is an error message to warn the user that there is very little memory left between the stack and the dictionary.

A6.1.2 **Disk Error Messages**

There are four disk errors : Read, Write, Select, and Protect.

Read: Disk Error: Block n This message indicates an error occurred while trying to read the disk. Possible causes are no disk in drive, a single-density formatted disk in a drive defined double-density or vice versa, or a disk with a bad sector.

********** IMPORTANT **********

WHEN THIS ERROR OCCURS, A BLOCK BUFFER HAS BEEN FILLED WITH WHATEVER DATA WAS READ. This feature is useful because if the error was due to a bad sector, the portion read can be fixed and rewritten; but otherwise IT SHOULD BE FORGOTTEN WITH **FLUSH EMPTY-BUFFERS.** (The FLUSH is to assure the rewriting of any proper data from **other** block buffers.)

********** IMPORTANT **********

Write: Disk Error: Block n indicates an error occurred while trying to write to the disk. Possible causes are the same as Read:, or the presence of a write-protect tab. When this error occurs, if it is due to a bad sector you may retry with FLUSH ; if this fails, try writing to a formatted backup disk instead. If the error was due to one of the other causes, fix the error and type FLUSH .

Select: Disk Error: Block n This error is caused by referencing a block which is higher than the highest defined in your system.

Protect: Disk Error: Block n This error is caused by trying to write to a block whose number is less than the current value of PBLK. If you really want to write this block reset PBLK and type FLUSH .

A6.2 SYSTEM LOCK-UPS, FORCED REBOOTS AND NEAR-MISSES

MMSFORTH is a relatively forgiving environment with simple and effective error comments. However, one of the more puzzling introductory aspects of Forth is the ability to "lock up" the entire system with no cursor. In this event, the disk system programmer can simply reboot; in a few seconds the system will be back to debug his/her program or to avoid the improper entry.

Sometimes the disk user would like to avoid reloading a large and slow-loading program, or the tape user won't want to wait several minutes to reboot. There may be a faster way. Usually the Forth program is still in RAM and, if you are using a TRS-80, THERE IS A DIRECT WAY TO RECOVER IT. Boot your TRS-80 into Level II BASIC. (If you are using an Expansion Interface Unit or a Model III with disks, be sure to hold down the Break key while pressing the Reset button.) Press Enter, then enter SYSTEM , finally enter /19200 (that's the entry point of MMSFORTH). If you're lucky, you're back in business! With an Expansion Interface or Model III disk, the first lines of the current block will be corrupted; use an EMPTY-BUFFERS to ignore it.

The IBM PC does not support any equivalent manuever.

The ability to lock up the system is not an accident of Forth design. Rather, it is the result of giving the programmer unusual access to the computer itself - one of the strongest features of Forth. The programmer is left to provide further limitations when desired for a particular task. In the absence of limitations, the lock-up can occur in several ways. Usual causes are "blowing the dictionary" or asking for something in a way the computer understands differently.

A6.2.1 Example (Blowing the Dictionary):

You are having a marvelous time because you have just discovered loop constructs. You write the simple routine:

```
: TROUBLE   BEGIN  13   0 UNTIL ;
```

As soon as you try it out, you hang the system and even pressing the Break key is of no use. What hit?

Your computer's available RAM may be large, but it never is infinite. You have just found a very efficient way to overwrite all available memory and then some by creating a rapidly growing stack with an insatiable appetite. After it fills all descending bytes of open space with single-precision 13's, it sails on down into the top of the dictionary with obviously disastrous results. Congratulations, you've created a monster!

In this particular case, you can even examine the damage your monster has wreaked. Do a **warm restart** (press Reset without powering down), then enter:

```
HERE  2000 DUMP
```

See all those single-precision (two-byte) 13's which were packed in at your bidding? Other bad things could go wrong, too. So from now on, be good! After crashing in this manner on an important operation, we recommend a **cold restart**; i.e., after powering down for at least 30 seconds.

A6.2.2 **Example (Nearly Blowing the Stack):**

You have just LOADed the appropriate blocks to run the SORT demo on your 16K MMSFORTH System Tape. Then you press Break, try a Forth routine from screen, and reLOAD the SORT blocks. You get a Dictionary full message. What happened?

If you had exited the SORT program by entering STOP from its menu screen, the program's exit code would have executed a FORGET TASK to remove the SORT code from the Forth Dictionary. After pressing the Break key you could have done the same. But instead you retained this program, which nearly fills available space in your 16K RAM; then you attempted to reLOAD. Wrong! To LOAD is to compile into Dictionary, and no way do you have room for a second complete set of SORT words in your tightly-packed RAM! The LOAD proceeds and the Dictionary grows up into higher RAM, until it threatens to eat up the last remaining bytes where the User Stack is growing down and to chew on into it. Before the stack and system are blown, MMSFORTH saves the day with its "Dictionary full" error message!

FORGET TASK, and run the underlying program.

A6.2.3 **Example (Simple Misunderstanding Between Friends):**

You attempt to copy a block to Block 40, but later discover the old contents in that block. Still later, you find some unexpected material in Block 64. What's wrong?

Most probably you were in HEX (Base 16) rather than DECIMAL mode. So the block was written to Hex 40, which is Decimal 64. Copy it into Block 40. If the old Block 64 was important, copy it back from a backup; or start typing! This time, remember to say DECIMAL before it's too late!

A6.2.4 **Example (Invisible Ink on IBM PC):**

Can't see what you are typing? SET-COLOR permits you to set background and foreground colors equal, but that doesn't mean you'll like it! 0 7 SET-COLOR resets white on black.

A6.2.5 **Example (Unusual Display Mode on IBM PC):**

You just finished editing a block, and the computer hasn't returned to a full screen display. What's worse, you've typed into the lower right corner of the small display, and only the final letter can now be changed!

That will remind you to exit the Editor with an Alternate-Q, not a Break! Press Enter, then reset the normal SET-WINDOW attributes with W/0 PAGE and Enter.

A6.3 **AMNESIA**

As our term suggests, in this case the system doesn't remember much of anything. You may be in the wrong Vocabulary - enter FORTH and try again. If the system returns FORTH ? you have blown the Dictionary. Congratulations, there is no recovery. You have messed up the address pointers which are essential to Forth's indirect threaded code (ITC) and the lower words can no longer be found. Re-boot!

Probable causes are storing past the limits of an array, storing a string that is longer than the maximum length of its $VARIABLE, or using bad limits on CMOVE, <CMOVE, FILL, BLANK, ERASE, etc.

To prevent the first, check the number of operations vs. size of array, and which value comes first on stack. To prevent the next, force a proper LEFT$ truncation before storing the string. If the others give you trouble, you can debug by temporarily adding a dummy word instead, such as:

```
: MOVEX   MOVE  . . . QUIT ;
```

Use this to do the operation and to examine what is on stack at that point.

A6.4 AND OTHERS

Resourceful Forth programmers can find other ways to cause these and related troubles. To mention a few:

1. Pressing Shift-0 on a TRS-80 Model I which does not have lowercase installed. (Repeat the operation to escape).

2. Mixing up @ and C@, ! and C!, etc.

3. Inserting an inappropriate stack operation between a paired >R and R> .

4. Running a 32K, 48K, or 64K RAM-sized program on a smaller-sized MMSFORTH System. You can check available RAM space before and after loading, with **'S PAD - U.** , or by using the **.MEM** word in TOOLKIT.

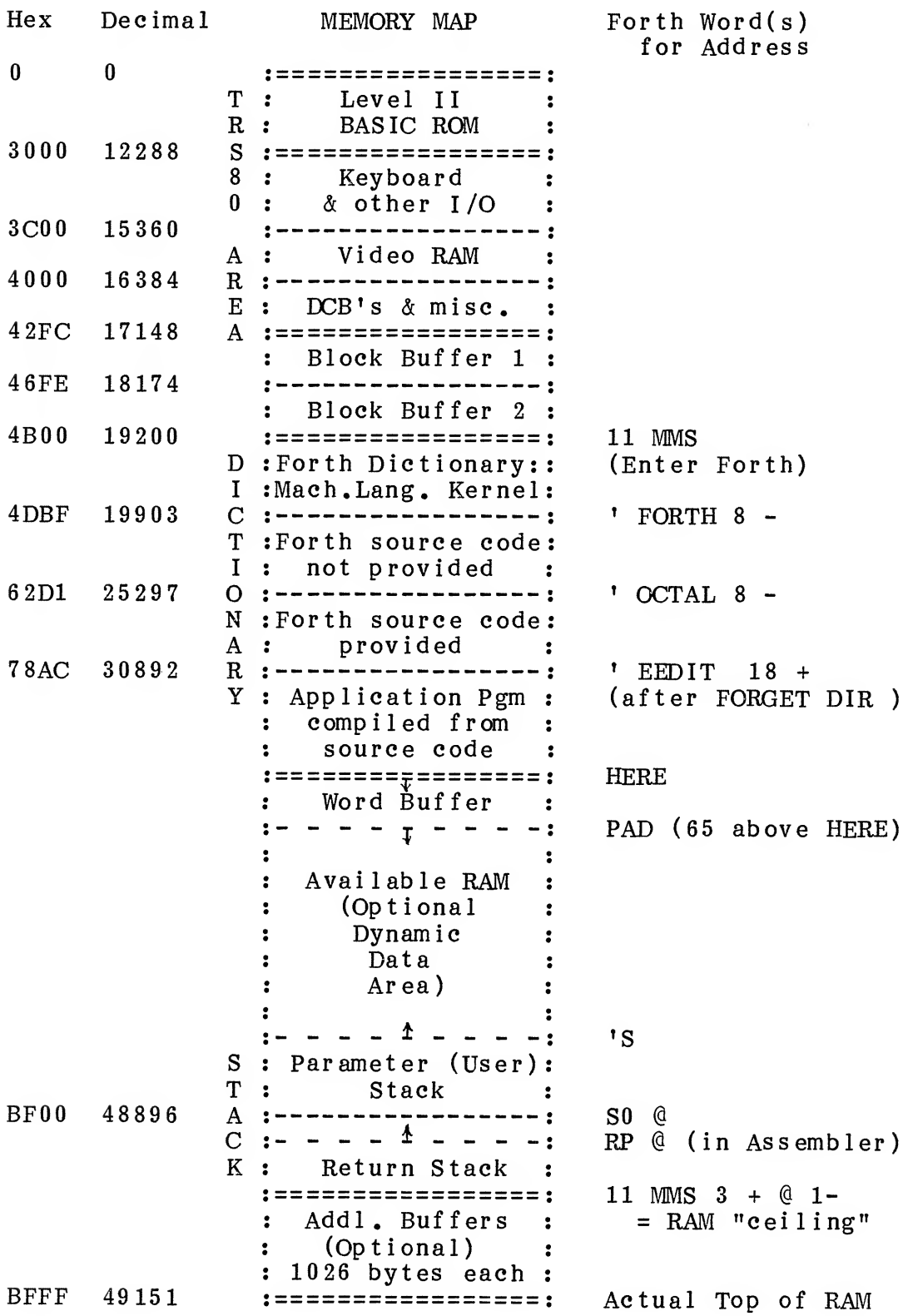
5. Permitting two lines of Forth code to accidentally run together. Remember that your editing must account for multiple-line wraparound during the LOAD operation.

A6.5 IT COULD BE HARDWARE

We hope not but it does happen - a particular bug in a RAM chip, poor circuit trace, etc., can look like a bug in one program only. If your program misbehaves identically on an equivalent computer (running at the proper RAM size), it's not a hardware problem.

One unusual but possible hardware incompatability concerns disk drives which, through design or misadjustment, take too long to come up to speed. Expect this if MMSFORTH backups successfully onto diskettes formatted by DOS but not onto those formatted by MMSFORTH. (The MMSFORTH FORMAT routine wrote the first bit of the sector header before the disk arrived at the right spot; later, other MMSFORTH routines cannot find a proper sector header.) CUSTOMIZE offers an easy way to increase MMSFORTH's disk start-up speed delay to compensate. Several other adjustments are possible - advanced users are referred to Appendix A12.

Figure 1 - **TYPICAL MMSFORTH MEMORY MAP** (V2.0, TRS-80 with 32K RAM)
(NOTE: To **display** addresses, use U.)



A8.0 **CATALOG LISTING OF MMSFORTH** (TRS-80)

A CATALOG listing of the current vocabulary may be made at any time in standard MMSFORTH. Just enter CATALOG, preceded by the vocabulary name (FORTH, EDITOR or ASSEMBLER) if required. You can pause the display with a Shift-@ and restart it again with any other printing key, or you can abort the display by pressing Break.

Your own CATALOG listings will show each word's number of characters (up to 31) immediately followed by an I if the word is to be immediately executed while compiling a definition, the first three characters of the word, and a character indicating the hash-code value of the following characters. If the word has exactly four characters, all four characters will be shown, instead. MMSFORTH shows a Dup-name report when compiling a definition whose CATALOG name coincides with another existing in the present dictionary.

The CATALOG listing on the following page has been modified to show entire wordnames but is in the usual order, starting from the top (the most recently added word). It also has had the RANDOM Extension wordset added to the over 300 words of MMSFORTH's basic FORTH vocabulary.

This listing has been enhanced with several lines and a box. The box surrounds the RANDOM wordset, demonstrating that the process of calling RANDOM from the DIRectory did a FORGET DIR to remove the DIRectory wordset, loaded the required blocks, and then replaced the DIRectory wordset on top by calling DIR itself before completing the action. Calling DIR after forgetting it is possible because the forgotten word was a **Dup-name** and the lower definition still exists immediately above (to the left on this CATALOG) DIRBLK. This first definition of DIR is simply DIRBLK LOAD so calling it recompiles the DIRectory words into the dictionary. The Auto-command in CUSTOMIZE should be set to DIR in order to accomplish this on boot-up, as on your original System Diskette (whose precompiled dictionary stopped at EEDIT).

The other lines across this CATALOG listing show which words are deleted in common RAM-saving calls: FORGET *, FORGET DIRBLK, and FORGET OCTAL. OCTAL is the lowest word for which source code is provided, so we FORGET OCTAL before reprecompiling basic MMSFORTH if modifications are made in Blocks 16-39. Block 16 requires that we shift into HEX, so we must say HEX **before** saying FORGET OCTAL - once OCTAL is forgotten, HEX will not be in the system either!

The CUSTOMIZE Utility resets a few parameters and then rewrites the lowest blocks on your Drive 0 with the present version of compiled code - including whatever you have compiled into the dictionary since booting. CUSTOMIZE also does a FORGET DIR so be sure to have a second DIR aboard or it will forget the lower DIR instead, probably an error which will require starting again! Once you have learned to use CATALOG, it should be scanned before using CUSTOMIZE to see that your dictionary contains just what is intended.

A8.1 **CATALOG LISTING OF FORTH VOCABULARY** (TRS-80)

```
CATALOG

 9  CHECKBOOK  7  NOTEPAD   10  BREAKFORTH 5  ALIFE      4  LIFE
 5  SORTS      5  $SORT      5  GUESS      8  PROGRAMS   9  CUSTOMIZE
 7  ALLCAPS    9  TRANSLATE  6  SEARCH     6  COPIES     6  BACKUP
 6  FORMAT     9  UTILITIES  7  TOOLKIT    5  CLOCK      8  CASSETTE
12  SCREEN-PRINT  GRAPHICS   6  RANDOM     7  STRINGS    6  ARRAYS
 8  DBL-PREC  10  EXTENSIONS 4  +FOR       3  +IS        2  ->
 5  MODEL      3  DIR        9  RANDOMIZE  3  RFR        3  RND
 3  RN1        4  SEED       7  MODULUS    3  MUL        5  EEDIT
 1  E          4  EDIT       6I EDITOR    11 79-STANDARD   MOVE
 7  CONVERT   12  SAVE-BUFFERS  CATALOG    5  .NAME      5  INDEX
 6  PLISTS     5  PLIST      5  BLIST      1  L          4  LIST
 2  TL         3  ?TL        4  DUMP       4  COPY       3  SCR
 4  OUTP       3  INP        2  D-         5  2OVER      5  2SWAP
 4  ROLL       4  PICK       5  */MOD      2  */         2  M/
 1  /          3  MOD        5  M/MOD     4  /MOD       2  M*
 1  *          2  ?S         2  C?        1  ?          2  U.
 3  U.R        5  DEPTH      2  'S         2  :3         2  :2
 2  :1         2  :0         5  FLUSH     13 EMPTY-BUFFERS  :R
 4  PCRT       3  CRT        5  PRINT      6  MARGIN     3  PTC
 9  -TRAILING  2  2/         2  OR         3  MAX        3  MIN
 3  NOT        5  BLANK      5  ERASE      4  FILL       6I CASEND
 9I OTHERWISE  5I NCASE      5I ACASE      3  DIR        6  DIRBLK
 4  BOOT       4  DDEN       4  SDEN       5  LEAVE      2I DO
 5I +LOOP      4I LOOP       4  PAGE       1  J          2  I'
 1  I          2  .R         6  SPACES     3  Y/N        3  #IN
 5  ENTER      6  <CMOVE     5I ;CODE      4  CODE       5  LABEL
 9I ASSEMBLER  6  BUFFER     9  CVARIABLE  8  VARIABLE  DEFINITIONS
10  VOCABULARY 2I ."         2I ("         6  FORGET     1I '
 9I [COMPILE]  5  ALLOT      6I MYSELF     7I LITERAL    4I ELSE
 6I REPEAT     4I THEN       5I WHILE      2I IF         5I UNTIL
 5I BEGIN      1I [          1  ]          9  IMMEDIATE  3  HEX
 7  DECIMAL    5  OCTAL      1  :          1I ;          4  FIND
 1I (          9  CCONSTANT  8  CONSTANT   6  CREATE     4  LOAD
 5  LOADS      6  NUMBER     3  MMS        5I DOES>      7  COMPILE
 8  QUESTION   3  -->        4  WORD       5  QUERY      6  EXPECT
 5  BLOCK      6  BUFFER     7  CURRENT    7  CONTEXT    2  S0
 2  DP         2  UT         3  >IN        3  BLK        6  CURSOR
```

3	#PT	3	HI#	4	EPOS	4	EBLK	5	STATE
4	LAST	4	#BKS	4	WBLK	4	RBLK	3	D?#
7	DWTSECS	7	DRDSECS	1	.	3	(.)	2	#S
1	#	4	SIGN	2	#>	2	<#	4	HOLD
5	SPACE	2	CR	4	BASE	2	BL	4	PBLK
4	QUIT	5	ABORT	1	1	1	0	2	-1
2	2-	2	1-	2	2+	2	1+	2	>R
2	R>	1	,	2	C,	4	HERE	4	SWAP
3	ROT	4	OVER	4	?DUP	3	DUP	4	DROP
2	C!	2	C@	2	+!	1	!	1	@
2	0>	2	0<	2	0=	2	U<	2	>=
1	>	2	<>	1	=	2	<=	1	<
3	AND	3	XOR	1	-	1	+	3	ABS
6	NEGATE	2	M+	2	D+	7	>BINARY	8	DISKDATA
8	BUFFDATA	2	U*	3	D0=	5	2DROP	4	2DUP
7	DNEGATE	5	TOKEN	3	16*	2	8*	2	2*
3	64*	3	PAD	4	EMIT	4	TYPE	5	COUNT
2	R@	5	U/MOD	3	KEY	4	?KEY	5	CMOVE
6	UPDATE	7	EXECUTE	5	%CONT	4	EXIT	5I	FORTH

A9.0 **MMSFORTH GLOSSARY** (FORTH VOCABULARY)

NOTES:

1. The following glossary is presented in order of ascending ASCII code. In general, it excludes words which appear in the MMSFORTH ASSEMBLER vocabulary. It includes key words from the FORTH vocabulary extensions.

2. The dictionary vocabulary of each FORTH word is listed, as well as the screen number in which it is defined, the number and type of stack entries its operation will "take in" and the number and type of entries it will return to stack.

3. A "Screen 0" listing implies that the word is precompiled within TRS-80 Blocks 0-11, IBM PC Blocks 0-16, or MMSFORTH Tape System Blocks 1-12, but source code is not supplied.

4. The first line of each Glossary entry contains the name of the word and a coded list of attributes in the following order.

   a) The vocabulary or extension:
   FORTH, **AS**SEMBLER, **ED**ITOR, **D**BL-**P**REC, **ST**RINGS, **CA**SSETTE, **GR**APHICS, **RA**NDOM, **AR**RAYS, **TO**OLKIT, **CL**OCK.
   b) The 79-STANDARD Word Set to which it belongs:
   STANDARD, **EX**TENSION, **NO**T-STANDARD.
   c) The attributes of the word:
   IMMEDIATE, **CO**LON-DEFINITION, and/or **U**SER **V**ARIABLE;
   d) The block number where the source code may be found (TRS-80/IBM PC, or 0 if source code is not provided).
   e) The status with regard to the earlier MMSFORTH version 1.9:
   NEW, **CH**ANGED, **UN**CHANGED.
   f) How to pronounce the word (if questionable).
   g) The stack before and after using the word.
   Codes used in this portion are as follows:
      1) addr - a value representing the single-precision address of a field.
      2) daddr - a value representing the double-precision address of a field. (IBM PC only).
      3) byte - a value representing an 8-bit byte. When in a larger field, the higher bits are zero.
      4) char - a value representing a 7-bit ASCII character code.
      5) d - a 32-bit signed 'double' number. The most significant 16-bits, with sign, is on TOS. (NOTE: This is **opposite** the way it was done in MMSFORTH V.1.9.)
      Values may be from -2,147,483,648 to 2,147,483,647.

6) ud - a 32-bit unsigned 'double' number.
7) flag - a numerical value with two logical states; 0 = false, non-zero = true.
8) n - a 16-bit signed integer number in the range -32,768 to 32,767.
9) un - a 16-bit unsigned integer number.
10) $ - a string. All strings are held in memory as a one-byte length count immediately followed by the string. When a string is "put on the stack" only the address of the length byte is actually placed on the stack.
11) <name> - An arbitrary FORTH word accepted from the input stream.
12) <namex> - an arbitrary FORTH word defined by <name>.

5. As a user convenience, a short-form listing of the abbreviations under 4.a-e above is printed at the bottom of each pair of pages.

6. 79-STANDARD source code may be loaded **into** the MMSFORTH System without problem. However, MMSFORTH words which are not in the 79-STANDARD subset are **not** designed for portability to **other** systems and you have agreed not to misuse the MMSFORTH source code by transferring it. You **can** use MMSFORTH to write 79-STANDARD programs for portability: LIMIT THE PROGRAM'S MMSFORTH VOCABULARY TO 79-STANDARD WORDS, referencing this Glossary or the FORTH 79-STANDARD MANUAL to confirm the appropriateness of your word set.

7. Words are arranged in ASCII code order as follows:
! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\ ] ^ _ ` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~

! FO,ST,,0,UN,"store",(n addr ->)
Stores number into address.
0 PBLK ! zeroes contents of variable named PBLK.

FO,ST,,0,CH,"number-sign",(ud1 -> ud2)
Converts the least significant digit of an unsigned 32-bit binary number to its ASCII equivalent using the current base. The ASCII character is placed in the output string. This word works by dividing the double-precision number on the stack by the base. The remainder is converted to ASCII and the quotient is left on the stack for further processing.
This command is somewhat equivalent to # in Level II BASIC's PRINT USING command.

```
: T# DUP ABS 0 <# # # 46 HOLD #S ROT SIGN #>
    TYPE ;
```

n T# will print n with a sign (if negative) and two digits after the decimal point.

#> FO,ST,,0,UN,"number-sign-greater",(ud -> addr n)
Ends pictured numeric output conversion. Drops ud, leaving the byte count of the just created text string in top-of-stack and its address beneath, suitable for use by TYPE .
See # (above) for example and further explanation.

#BKS FO,NO,UV,0,NE,"number-b-k-s",(-> addr)
Leaves address of character variable containing the current number of available block buffers. Use CUSTOMIZE to reset the current number of block buffers. To readjust the maximum number of block buffers, modify Block 15 and recompile the system.

#CHRS FO,NO,,0,NE,"number-characters",(-> n) (IBM PC only)
Returns number of character spaces remaining from current cursor position to end of line in current video window.

#IN FO,NO,,21/38,UN,"number-in",(-> n)
Outputs " ? " and inputs a signed integer number (no decimal point) and puts the low order 16 bits on the stack. The high-order 16 bits are stored in HI#. Included blank characters or other bad input returns ? Redo, and prompts for more input.

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

#PT FO,NO,UV,0,UN,"number-p-t",(-> addr)
1-byte variable containing position of decimal point in last number input:
0 if no decimal point
1 if decimal point is last character of number
2 if 1 digit follows decimal point, etc.

#S FO,ST,,0,CH,"number-sign-s",(ud -> 0.)
Converts all digits of an unsigned 32-bit (double-precision) binary number on the stack, to their ASCII equivalents, using the current base. The ASCII characters are placed in the output string. At least one digit will be converted if the number is 0. Use only between <# and #>. (See # .)

$! ST,NO,,50/69,UN,"string-store",($ addr ->)
Move the $ to addr for the length of the string.
WARNING: If the string at $ is longer than the space available at addr, it will overflow and may destroy the dictionary. Therefore, if in doubt, use LEFT$ to truncate the string to the correct size.
See Chapter 8, the CHECKBOOK program for usage.

$" ST,NO,,50/69,CH,"string-quote",(-> $)
Creates a string literal of following characters to " , leaves its address on the stack. If in a colon-definition, the literal is placed in-line in the dictionary; if in interpretive mode, the literal is placed in PAD + 256.

$+ ST,NO,,51/70,UN,"string-concatenate",($1 $2 -> $)
String concatenation operator. The string at $2 is appended to the string at $1 and the result is left in PAD. Leaves address of PAD.

$-TB ST,NO,,52/71,UN,"string-minus-t-b",($ -> $)
Removes trailing blanks from a string whose address is at top-of-stack.

$. ST,NO,,50/69,UN,"string-dot",($ ->)
Outputs string whose address is at top-of-stack.

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

$ARRAY ST,NO,,52/71,UN,"string-array",(n1 n2 ->)
A defining word used in the form:

```
n1 n2 $ARRAY <name>
```

to create a dictionary entry for <name>, consisting of a 1 byte minimum string length n1 (maximum 254) +1 and n2+1 elements (indexes 0 to n2), each of length n1+1. The entire array is cleared to zeroes.
When <name> is later used, it is in the form:

```
n <name>
```

where the address of element n is put on the stack.

```
20 5 $ARRAY AR ok
```

defines a string array of six elements, where each string is 20 characters plus the length byte long.
To initialize element three:

```
$" THREE" 3 AR $! ok
```

To retrieve and print element three:

```
3 AR $. THREE ok
```

$COMPARE ST,NO,,51/70,UN,"string-compare",($1 $2 -> flag)
Compare two strings.
Flag returned is as follows:

```
If $1=$2, f=0.
If $1<$2, f=-1.
If $1>$2, f=+1.
```

$CONSTANT ST,NO,,50/69,UN,"string-constant"
A defining word used in the form:

```
$CONSTANT <name> ..."
```

to create a dictionary entry for <name>, consisting of the string starting one space after <name> and terminated at the " .
When <name> is later used the string address of the constant string is placed on the stack.

```
$CONSTANT C1 ABC"
C1 $. ABC ok
```

The only difference between a string constant and a string variable is that the string constant is initialized and the length is set to the length of the initial value. There is no difference is use. Both return the address of the length byte when invoked.

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

$T CL,NO,UV,60/79,NE,"string-t",(-> addr)
Variable in which is stored the title for TLISTS and TINDEX.

$VARIABLE ST,NO,UV,50/69,UN,"string-variable",(n ->)
A defining word used in the form:
n $VARIABLE <name>
to create a dictionary entry for <name>, consisting of n+1 bytes (maximum 255) which are all initialized to zero.
When <name> is later used the string address is placed on the stack. n is the maximum size string which can be stored in <name>.
The only difference between a string constant and a string variable is that the string constant is initialized and the length is set to the length of the initial value. There is no difference is use. Both return the address of the length byte when invoked.

$XCHG ST,NO,,50/69,UN,"string-exchange",($1 $2 ->)
Exchanges contents of the two strings, using PAD for temporary storage. The two strings should have the same maximum lengths.

%CONT FO,NO,,0,NE,"per-cent-continue" (TRS-80)
May be used to end words which are activated by the Break key or shift-control-*, so that execution continues normally. (Only other word which can be used is ABORT.)

' FO,ST,IM,17/22,UN,"tick",(-> addr)
If executing, leaves the parameter field address of the next word accepted from the input stream. If compiling, compiles this address as a literal; later execution will place this value on the stack. First searches the CONTEXT vocabulary, then the CURRENT vocabulary, before giving an error message. Can be used to find out if a word has been previously defined or to modify a constant. If CONST was defined as 0 CONSTANT CONST , then:
4 ' CONST ! CONST . 4 ok

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

'S FO,NO,,32/39,UN,"tick-s",(-> addr)
Places onto the stack the address of top-of-stack before execution of 'S .
: MEM-SIZE 'S PAD - 5 SPACES U. 5 SPACES ;
prints unused bytes of RAM available.

(FO,ST,IM,0,CH,"paren"
Begins a comment, which is terminated by) or end of block. It must be followed be a space but the) need not be preceded by a space. The comment is ignored by the system and may appear inside or outside a colon-definition.

(" FO,NO,IM,17/22,UN,"paren-quote"
Alternate to (, it allows a) to be embedded in a comment without closing it. " is used to close (" .
(" IMBEDDED CLOSE-PARENS (LIKE THIS) ARE NOW OK"

(.) FO,NO,,0,UN,"paren-dot-close-paren",(n -> addr n)
Converts n to its ASCII equivalent, leaving its address and byte count on the stack, suitable for use by TYPE .

(D.) DP,NO,,44/63,NE,"paren-d-dot-paren",(d -> addr n)
Converts d to its ASCII equivalent, leaving its address and byte count on the stack, suitable for use by TYPE .

***** FO,ST,,33/51,UN,"times",(n1 n2 -> n3)
Leaves the product of n1 times n2.
4 3 * . 12 ok

***/** FO,ST,,33/51,UN,"times-divide",(n1 n2 n3 -> n4)
Multiplies n1 by n2, then divides by n3. n4 is rounded toward zero. The product of n1 times n2 is maintained as a 32-bit value for greater precision.
100 4 3 */ . 133 ok

***/MOD** FO,ST,,33/51,UN,"times-divide-mod",(n1 n2 n3 -> n4 n5)
Multiply n1 by n2, divide the 32-bit result by n3 and leave the remainder n4 and the quotient n5. The remainder has the same sign as n1.
100 4 3 */MOD . . 133 1 ok

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

+ FO,ST,,0,UN,"plus",(n1 n2 -> n3)
Leaves the arithmetic sum of n1 plus n2.

+! FO,ST,,0,UN,"plus-store",(n addr ->)
Add n to the 16-bit value at the memory address on TOS.

+LOOP FO,ST,IM&CO,22/39,CH,"plus-loop",(n ->)
Used instead of LOOP when an increment (n) other than +1 is desired. The loop is terminated if the index equals or exceeds the limit when the increment is positive or is less than the limit when the increment is negative. Index and limit are signed integers in the range -32,768 to 32,767.

+TINDEX CL,NO,,60/79,NE,"plus-t-index",(n1 n2 ->)
Does a TINDEX without resetting TPAGE to 1.

+TLISTS CL,NO,,60/79,NE,"plus-t-lists",(n1 n2 ->)
Does a TLISTS without resetting TPAGE to 1.

, FO,ST,,0,UN,"comma",(n ->)
Compiles n into the next two bytes in the dictionary at HERE and increments HERE by two.

- FO,ST,,0,UN,"minus",(n1 n2 -> n3)
Subtracts n2 from n1 and leaves difference n3.

```
123 23 - . 100  ok
```

--> FO,NO,IM,0,NE,"next-screen"
During a screen load, ignores following words on the current screen, sets UT to 1 to load **one** following screen via same block buffer.

-1 FO,NO,,0,UN,"minus-one",(-> -1)
Pushes -1 onto the stack (shortens dictionary search).

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

-TRAILING FO,ST,,26/40,UN,"minus-trailing",(addr n1 -> addr n2)
Reduces the character count (in n1) of the string at addr by the number of trailing blanks (blanks to the right of the string), leaving the new character count n2.

. FO,ST,,0,CH,"dot",(n ->)
Outputs n converted according to BASE followed by one blank. Only a negative sign is displayed.

." FO,ST,IM,17/22,CH,"dot-quote"
Accepts following text until " or end of block for the input stream. If executing, outputs immediately. If compiling, compile so text will be output on execution.

```
: GREETING PAGE 10 25 PTC ." HI THERE!" CR ;
    GREETING
or PAGE 10 25 PTC ." THIS GOES OUT RIGHT AWAY" CR
```

.NAME FO,NO,,36/54,NE,"dot-name",(addr ->)
Given addr of a name field in the dictionary, outputs the length, I if immediate, the 1st three characters of the name, and an interpretation of the hash code.

.MEM TO,NO,,61/80,NE,"dot-mem"
Outputs the number of bytes between PAD and 'S.

.R FO,NO,,21/40,UN,"dot-r",(n1 n2 ->)
Outputs n1 according to base, right-justified in a field whose width is n2. If n2 is smaller than the characters required for n1, no leading spaces are given.

```
: GRAPHICS CR 64 0 DO I 128 + DUP 6 .R SPACE
    EMIT LOOP CR ;
```

.S TO,NO,,61/80,NE,"dot-s",(... -> ...)
Output the contents of the user stack without altering those contents. Displays TOS on right.

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

/ FO,ST,,33/51,UN,"divide",(n1 n2 -> n3)
Divides n1 by n2 leaving quotient n3. n3 is rounded toward zero.

```
5 2 / . 2  ok
```

/MOD FO,ST,,33/51,UN,"divide-mod",(n1 n2 -> n3 n4)
Divides n1 by n2 leaving remainder n3 and quotient n4. n3 has the same sign as n1.

```
5 2 /MOD . . 2 1  ok
```

0 FO,NO,,0,UN,"zero",(-> 0)
Pushes 0 onto the stack (shortens dictionary search).

0. DP,NO,,43/63,NE,"zero-dot",(-> d0)
Pushes double number 0 onto the stack (shortens dictionary search).

0< FO,ST,,0,UN,"zero-less",(n -> flag)
True if n is less than zero (i.e., negative).

0= FO,ST,,0,UN,"zero-equals",(n -> flag)
True if n is equal to zero.

0> FO,ST,,0,UN,"zero-greater",(n -> flag)
True if n is greater than zero (i.e., positive).

0C! FO,NO,,49,NE,"zero-c-store",(byte addr ->) (IBM PC only)
Stores byte in addr in segment 0.

0C@ FO,NO,,49,NE,"zero-c-fetch",(addr -> byte) (IBM PC only)
Fetches byte from addr in segment 0.

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

0TPAGE CL,NO,,60/79,NE,"zero-t-page"
Reinitializes TPAGE counter to 1 for TLISTS and TINDEX.

1 FO,NO,,0,UN,"one",(-> 1)
Pushes 1 onto the stack (shortens dictionary search).

1+ FO,ST,,0,UN,"one-plus",(n -> n+1)
Increments n by 1.

1- FO,ST,,0,UN,"one-minus",(n -> n-1)
Decrements n by 1.

16* FO,NO,,0,UN,"sixteen-times",(n -> n*16)
Multiplies n by 16.

2! DP,EX,,43/62,CH,"two-store",(d addr ->)
Stores d at addr for four bytes. High-order portion of double number is in first word and is higher on stack.

2$ARRAY ST,NO,,52/71,UN,"two-string-array",(n1 n2 n3 ->)
A defining word used in the form:

```
n1 n2 n3 2$ARRAY <name>
```

to create a dictionary entry for <name>, consisting of 1 byte for the high-row-index n2 (maximum 254) +1, 1 byte for the maximum-string-length n1 (maximum 254) +1 and n3+1 columns (indexes 0 to n3) of n2+1 rows (indexes 0 to n2), each of length n1+1. The entire array is cleared to zeroes.
When <name> is later used it is in the form:

```
n4 n5 <name>
```

where the string address of element n4,n5 is put on the stack.
Define a two-dimensional string array named Q1, of 8 columns by 6 rows with 20-byte strings:

```
20 5 7 2$ARRAY Q1
```

Set Element 1,5 equal to Element 2,7 :

```
2 7 Q1  1 5 Q1 $!
```

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

2* FO,NO,,0,UN,"two-times",(n -> n*2)
Doubles n.

2+ FO,ST,,0,UN,"two-plus",(n -> n+2)
Increments n by two.

2- FO,ST,,0,UN,"two-minus",(n -> n-2)
Decrements n by two.

2/ FO,NO,,26/0,UN,"two-divide",(n -> n/2)
Halves n.

2@ DP,EX,,43/62,CH,"two-fetch",(addr -> d)
Leaves the contents of the four consecutive bytes beginning at addr on the stack as a double number.
The first word is the high-order half and is on the top of stack.

2ARRAY AR,NO,,49/68,UN,"two-array",(n1 n2 ->)
A defining word used in the form:

```
n1 n2 2ARRAY <name>
```

to create a dictionary entry for <name>, consisting of the 1 byte high-row-index n1 (maximum 254) +1, and n2+1 columns (indexes 0 to n2) of n1+1 rows (indexes 0 to n1) of two-byte elements.
When <name> is later used it is in the form:

```
n3 n4 <name>
```

where the address of Element n3,n4 is put on the stack.
Define a two-dimensional array named A1, 8 columns by 6 rows.

```
5 7 2ARRAY A1
```

Set Element 1,5 equal to 7:

```
7 1 5 A1 !
```

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

2CARRAY AR,NO,,49/68,UN,"two-c-array",(n1 n2 ->)
A defining word used in the form:
n1 n2 2CARRAY <name>
to create a dictionary entry for <name>, consisting of the 1 byte high-row-index n1 (maximum 254) +1, and n2+1 columns (indexes 0 to n2) of n1+1 rows (indexes 0 to n1) of one-byte elements.
When <name> is later executed it is executed in the form:
n3 n4 <name>
to put the element address of Element n3,n4 on the stack.
Define a two-dimensional byte array named A1, 8 columns by 6 rows.

```
5 7 2CARRAY A1
```

To set Element 1,5 equal to 7:

```
7 1 5 A1 C!
```

2CONSTANT DP,EX,,43/62,CH,"two-constant",(d ->)
A defining word used in the form:
d 2CONSTANT <name>
to create a dictionary entry for <name>, consisting of the 4-byte number d.
When <name> is later executed the value d is put on the stack.

```
5. 2CONSTANT FIVE
```

2DARRAY AR,NO,,49/68,CH,"two-double-array",(n1 n2 ->)
A defining word used in the form:
n1 n2 2DARRAY <name>
to create a dictionary entry for <name>, consisting of the 1 byte high-row-index n1 (maximum 254) +1, and n2+1 columns (indexes 0 to n2) of n1+1 rows (indexes 0 to n1) of four-byte elements.
When <name> is later used it is in the form:
n3 n4 <name>
where the address of Element n3,n4 is put on the stack.
Define a two-dimensional array named A1, 8 columns by 6 rows.

```
5 7 2DARRAY A1
```

Set Element 1,5 equal to 7:

```
7. 1 5 A1 D!
```

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

2DROP DP,EX,,0,UN,"two-drop",(d ->)
Drops top double number from stack.

2DUP DP,EX,,0,UN,"two-dupe",(d -> d d)
Duplicates the top double number on the stack.

2EDIT TO,NO,,61/80,NE,"two-edit",(n ->)
Diagnostic tool to determine equivalence of two blocks at same relative location on two disks. Reads Block n on Drives 0 and 1 into Block Buffers 0 and 1. Use Alternate-E (Shift-Clear-E on TRS-80) to Exchange the two blocks on screen. Any differing code will flicker.

2EDITS TO,NO,,61/80,NE,"two-edits",(n1 n2 ->)
Does 2EDIT on consecutive blocks from Block n1 for a count of n2 blocks. Edit and Alternate-S or Alternate-Q (Shift-Clear-S or Shift-Clear-Q on TRS-80) to proceed to next pair of blocks.

2OVER DP,EX,,34/40,UN,"two-over",(d1 d2 -> d1 d2 d1)
Copies the second double number onto the stack.

2ROT DP,EX,,44/63,UN,"two-rote",(d1 d2 d3 -> d2 d3 d1)
Rotates the third double number to the top of the stack.

2SWAP DP,EX,,34/40,UN,"two-swap",(d1 d2 -> d2 d1)
Exchanges the top two double numbers on the stack.

2VARIABLE DP,EX,,43/63,CH,"two-variable"
A defining word used in the form:

```
2VARIABLE <name>
```

to create a dictionary entry for <name>, consisting of four bytes.
When <name> is later used the PFA of <name> is put on the stack.

```
2VARIABLE FIVE  5. FIVE D!
```

64* FO,NO,,0,UN,"sixty-four-times",(n -> n*64)
Multiplies n by sixty-four.

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

79-STANDARD FO,ST,,36/54,NE,"seventy-nine-standard"
A null word which allows the system to be checked to see if FORTH 79-STANDARD is supported.

8* FO,NO,,0,UN,"eight-times",(n -> n*8)
Multiplies n by eight.

: FO,ST,,0,UN,"colon",(: <name> . . . ;)
Defining word used to create a dictionary entry for <name> in CURRENT vocabulary, which is called a colon-definition. CONTEXT is set to equal CURRENT and STATE changed to compilation. The compilation addresses of subsequent words from the input stream which are not immediate words are stored into the dictionary to be executed when <name> is later executed. IMMEDIATE words are executed as encountered. If a word is not found after a search of the CONTEXT and CURRENT vocabularies, conversion and compilation as a literal number is attempted, with regard to the current BASE. Failing that, an error condition exists.

:0 FO,NO,,32/47,NE,"colon-zero",(n1 -> n2)
Converts relative block number n1 to absolute block number n2 for Drive 0.

:1 FO,NO,,32/47,NE,"colon-one",(n1 -> n2) Converts relative block number n1 to absolute block number n2 for Drive 1.

:2 FO,NO,,32/47,NE,"colon-two",(n1 -> n2)
Converts relative block number n1 to absolute block number n2 for Drive 2.

:3 FO,NO,,32/47,NE,"colon-three",(n1 -> n2)
Converts relative block number n1 to absolute block number n2 for Drive 3.

:R FO,NO,,32/47,NE,"colon-r",(n1 n2 -> n1 n3)
Converts a range of numbers, n1 through n2, to starting n1, and count n3.

; FO,ST,IM&CO,0,CH,"semi-colon"
Terminates a colon-definition and sets STATE to Execute.

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

;CODE AS,EX,IM&CO,20/37,UN,"semi-colon-code"
(: <name> ... ;CODE ...)
Ends the creation portion of a new defining word and begins the execution portion) of it after setting the CONTEXT vocabulary to ASSEMBLER and STATE to Execute.
When <name> is executed in the form: <name> <namex> to define <namex>, the compilation address of <namex> will contain the address of the code sequence following the ;CODE in <name>. Execution of any <namex> will cause this machine code sequence which follows ;CODE to be executed.

< FO,ST,,0,NU,"less-than",(n1 n2 -> flag)
True if n1 is less than n2.

<# FO,ST,,0,UN,"less-number-sign"
Initializes for "pictured" numeric output. (See #)

<= FO,NO,,0,UN,"less-than-or-equal",(n1 n2 -> flag)
True if n1 is less than or equal to n2.

<> FO,NO,,0,UN,"not-equal",(n1 n2 -> flag)
True if n1 is not equal to n2.

<CMOVE FO,NO,,21/38,NE,"reverse-c-move",(addr1 addr2 n ->)
Copies n bytes beginning at addr1 to addr2. The move procedes within the n bytes from high memory toward low memory.

= FO,ST,,0,UN,"equals",(n1 n2 -> flag)
True if n1 equals n2.

> FO,ST,,0,UN,"greater-than",(n1 n2 -> flag)
True if n1 is greater than n2.

>= FO,ST,,0,UN,"greater-than-or-equal",(n1 n2 -> flag)
True if n1 is greater than or equal to n2.

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

>B:D FO,NO,,47,NE,"to-b-colon-d",(block# -> rel-block# drive#) (IBM PC)
Given a block number, returns a relative block number on drive number.

>BINARY FO,NO,,0,NE,"to-binary",(d1 addr1 -> d2 addr2)
Converts to a value the text string beginning at addr1+1 with regard to BASE. The new value is accumulated into d1 producing d2. addr2 is the address of the first non-convertible character. (Same as CONVERT .)

>IN FO,ST,UV,0,NE,"to-in", (-> addr)
User variable containing the present character offset within the input stream, (0 to 1023).

>R FO,ST,CO,0,UN,"to-r",(n ->)
Transfer n to the return stack. Every >R must be balanced by a R> in the same control structure nesting level of a colon-definition.

? FO,ST,,32/47,UN,"question-mark",(addr ->)
Displays the number at address, using the format of . (dot). Equivalent to @ . .

?CLR FO,NO,,0,NE,"question-color",(->) (IBM PC)
If current video display is color board then wait for vertical retrace and turn video off. If b/w board do nothing. Removes hash from updating color screen.

?DUP FO,ST,,0,CH,"question-dup",(n -> n) or (n -> n n)
Duplicates n if it is non-zero, otherwise does nothing.

?KEY FO,NO,,0,UN,"question-key",(-> char)
Checks to see if a key is being pressed. Returns 0 if no key pressed, else ASCII value of key .

?S FO,NO,,32/47,UN,"question-s"
Outputs value of stack pointer.

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

?TL FO,NO,,35/53,NE,"question-t-l",(n1 n2 ->)
Outputs Line n1 of Screen n2 if it is all printable characters.

@ FO,ST,,0,UN,"fetch",(addr -> n)
Leaves on the stack the number contained at addr.

ABORT FO,ST,,0,NE
Clears the user and return stacks, setting execution mode. Returns control to the terminal.

ABS FO,ST,,0,UN,(n1 -> n2)
Leaves absolute value n2 of n1.

ACASE FO,NO,IM&CO,25/21,UN,(char ->)
Begins alphabetic case structure of the form:

ACASE ccc" <name> <name> <name> OTHERWISE ... CASEND

At compilation it is followed by one blank and a list of ASCII characters each of which matches an appropriate routine in the dictionary. The list of characters is terminated with a " followed by the matching list of routines. When executed compares char with each defined character and executes appropriate <name>.

ACASE ZAP" ZRTN ARTN PRTN OTHERWISE ." BAD" CR CASEND

If top item on stack = 65 (ASCII "A") then ACODE is executed.

If top item on stack is not equal to any of the defined characters, control passes to the word following OTHERWISE (if present) or CASEND .

ALLOT FO,ST,,16/0,UN,(n ->)
Adds n bytes to the parameter field of the most recently defined word.

AND FO,ST,,0,UN,(n1 n2 -> n3)
Performs bitwise logical AND operation on n1 and n2.

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

ARRAY AR,NO,,49/68,UN,(n ->)
A defining word used in the form:
n ARRAY <name>
to create a dictionary entry for <name>, consisting of n+1 two-byte elements (indexes 0 to n).
When <name> is later used, it is in the form:
n <name>
which the address of Element n is put on the stack.
Define an array named B with 16 elements.
15 ARRAY B
Set Element 5 equal to Element 7:
7 B @ 5 B !

ASC ST,NO,,52/71,UN,"ass-key",($ -> char)
Returns ASCII value of first character in the string.

ASSEMBLER FO,NO,IM,18/23,UN
Name of the 8080 Assembler vocabulary. When this name is executed, ASSEMBLER is established as the CONTEXT vocabulary.

B/W FO,NO,,50,NE,"b-slash-w",(->) (IBM PC)
Sets current video board to black-and-white (i.e., monochrome) and initializes black-and-white board.
(B/W&COLOR option.)

B:D> FO,NO,,47,NE,"b-colon-d",(rel-block# drive# -> block#) (IBM PC)
Given a relative block number and drive number, returns an absolute block number.

BASE FO,ST,UV,0,UN,(-> addr)
User variable containing the radix for current input-output number conversions on input or output. Allowable range is 2 to 70.
: BINARY 2 BASE ! ;

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

BEGIN FO,ST,IM&CO,16/21,CH
Used only in colon-definitions. Marks the beginning of an indefinite loop in the forms:
BEGIN ... flag UNTIL or
BEGIN ... flag WHILE ... REPEAT
A BEGIN ... UNTIL loop will be repeated until flag is true. A BEGIN ... WHILE ... REPEAT loop will be repeated until flag is false. The words after UNTIL or REPEAT will be executed when either loop is finished. flag is always dropped after being tested.

BL FO,NO,,0,CH,"b-l",(-> char)
Leaves ASCII blank value (32 decimal) on stack.

BLANK FO,NO,,26/40,UN,(addr n ->)
Fill an area of memory starting at addr for n bytes, with ASCII blanks (decimal 32).

BLIST FO,NO,,35/53,NE,"b-list",(n ->)
Outputs Screen n, formatted with block number at the top and line numbers on the left.

BLK FO,ST,UV,0,NE,"b-l-k",(-> addr)
User variable containing the number of the mass storage block being interpreted as the input stream. If the content is zero, the input stream is taken from the terminal.

BLOCK FO,ST,,0,UN,(n -> addr)
Leaves the address of the first byte of Block n. If the block is not already in memory, it is transferred from mass storage into whichever memory buffer has been least recently accessed. If the block occupying that buffer has been UPDATEd (i.e. modified), it is rewritten onto mass storage before Block n is read into the buffer.

BOOT FO,NO,,15/20,NE
Causes the system to reload from disk. This is the software equivalent of pressing the Reset button.

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

BOX FO,NO,,52,NE,(modulus height width row col ->) (IBM PC
Draws a box whose upper left corner is at row col. The box is of given height and width and its horizontal lines are marked every modulus characters.

BUFFDATA FO,NO,UV,0,NE,(n -> addr)
Array containing information about the buffers.
See Appendix A12.

BUFFER FO,ST,,0,NE,(n -> addr)
Obtains the next block buffer, assigning it to Block n. The block is not read from mass storage. If the buffer has been previously marked as UPDATEd, it is written to mass storage. The address left is the first byte within the buffer for data storage.
WARNING (V.2.0 only): Results cannot be predicted if Block n is already in a buffer.

C! FO,ST,,0,UN,"c-store",(n addr ->)
Stores the least significant 8 bits of n at addr.

C, FO,NO,,0,UN,"c-comma",(n ->)
Stores the low-order 8 bits of n at the next byte in the dictionary, advancing the dictionary pointer by one.

C= FO,NO,,38,NE,"cursor-equals", (IBM PC)
(b/w cursor-top btm color cursor-top btm ->)
Defining word for cursors. eg:
12 13 6 7 C= UNLN-CUR
defines a word " UNLN-CUR " which when used will set a 2 line high cursor for both the b/w or color card, whichever is active. This cursor will be at the bottom of the character.

C? FO,NO,,32/47,UN,"c-question",(addr ->)
Outputs low-order 8 bits at address.

C@ FO,ST,,0,UN,"c-fetch",(addr -> byte)
Leaves on the stack the contents of the low-order character (8 bits) at addr. The high-order bits are zeroed.

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

CARRAY AR,NO,,49/68,UN,"c-array",(n ->)
A defining word used in the form:
n CARRAY <name>
to create a dictionary entry for <name>, consisting of n+1 one-byte elements (indexes 0 to n).
When <name> is later used it is in the form:
n <name>
where the address of Element n is put on the stack.
Define an array named B with 16 elements.
15 CARRAY B
Set Element 5 equal to Element 7:
7 B C@ 5 B C!

CASEND FO,NO,IM&CO,25/21,UN,"case-end"
Ends ACASE or NCASE conditional structure.

CATALOG FO,NO,,36/54,CH
Outputs length, first 3 characters, and hash code of defined words in CONTEXT dictionary. (Four-letter words shown in full.) Immediate words have I printed after size. The last "word" listed may be garbage.

CCONSTANT FO,NO,,0,UN,"c-constant",(byte ->)
A defining word used in the form:
byte CCONSTANT <name>
to create a dictionary entry for <name>, consisting of the byte in its parameter field.
When <name> is later used, the byte will be left in the low-order 8 bits of the stack entry with the high-order 8 bits zeroed.

CHR$ ST,NO,,52/71,UN,"c-h-r-string",(char -> addr)
Takes an ASCII value from top-of-stack, puts its equivalent string into PAD and puts PAD's present address on the stack.

CMOVE FO,ST,,0,CH,"c-move",(addr1 addr2 n ->)
Move n (at least 1) bytes beginning at address addr1 to addr2. The contents of addr1 are moved first, proceeding toward high memory. (See <CMOVE).

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

CODE FO,EX,,20/37,UN
A defining word which begins a dictionary entry for the word following, and makes ASSEMBLER the CONTEXT vocabulary. Used in the form:

```
CODE <name> ... NEXT ( or PSH or PSH2 )
```

COLOR FO,NO,,50,NE,(->) (IBM PC - color video only)
Sets current video board to color and initializes color board. (B/W&COLOR option.)

COLOR? FO,NO,,22,NE,"color-question",(-> f) (IBM PC)
Returns true if current video window is on color board.

COMPILE FO,ST,CO,0,NE
When a word containing COMPILE executes, the 16-bit value following the compilation address of COMPILE is copied (compiled) into the dictionary. I.e.,

```
COMPILE DUP       will copy the compilation
                  address of DUP, and
COMPILE [ 0 , ]   will copy zero.
```

CONSTANT FO,ST,,0,UN,(n ->)
A defining word used in the form:

```
n CONSTANT <name>
```

to create a dictionary entry for <name>, consisting of n in its parameter field.
When <name> is later executed, the N will be left on the stack.

```
5 CONSTANT FIVE
```

Not obvious what is meant by N

CONTEXT FO,ST,UV,0,UN,(-> addr)
A user variable specifying the first vocabulary in which dictionary searches are to be made during interpretation of the input stream.

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

CONVERT FO,NO,,36/54,NE,(d1 addr1 -> d2 addr2)
Converts to a value the text string beginning at add1+1 with regard to BASE. The new value is accumulated into d1 producing d2. addr2 is the address of the first non-convertible character. (Same as >BINARY .)

COPY FO,NO,,35/53,CH,(n1 n2 ->)
Copies Block n1 to Block n2 by reading Block n1 into memory, changing its block number in memory, and marking it as UPDATEd.
WARNING: Results cannot be predicted if n2 is already in a buffer. Use FLUSH or EMPTY-BUFFERS before COPY to clear the buffers.

COUNT FO,ST,,0,UN,(addr -> addr+1 n)
Leaves the address and the character count of the text string beginning at addr. The first byte of the text string at addr must contain the character count n. Range of n is 0 to 255.
It could be defined as:

```
: COUNT DUP 1+ SWAP C@ ;
```

CR FO,ST,,0,UN,"c-r"
Outputs a carriage return. (Most standard output devices also do an automatic line feed.)

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

CREATE FO,ST,,0,CH
Creates a dictionary entry for <name> without allocating any parameter field memory. When <name> is subsequently executed, the addess of the first byte of <name>'s parameter field is left on the stack.
CREATE can be used by itself to, for example, create an initialized array-like variable:

```
CREATE ARR  6 , 7 , 8 ,
```

creates a dictionary entry called ARR with 6, 7, and 8 in the parameter field. When ARR is called, the address of the first byte of its parameter field is put on the stack.
CREATE can also be used inside a colon-definition. In this case its execution is delayed until the actual execution of the word defined with the colon-definition.
For example:

```
: VARIABLE  CREATE 0 , ;
```

uses CREATE to give a name to the variable and then allocates two bytes which are initialized to zero. When creating a variable, CREATE finds <name> after VARIABLE and uses it to set up the dictionary entry, then the rest of the definition is executed to initialize the parameter field with zero.
Later, when <name> is called, its address is placed on the stack. Often used with DOES>.
This third level effect is quite abstract, like "thinking about thinking about thinking".

CRESET FO,NO,,0,NE,"color reset",(->) (IBM PC)
Turns color video back on. Used after ?CLR.

CRT FO,NO,,32/40,UN,"c-r-t",(->)
Makes the video display (**C**athode **R**ay **T**ube) the current output device.

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

CUR-POS F0,NO,,0,NE,"cursor-position",(-> addr) (IBM PC)
returns starting address for Cursor Position Table:

+0	column number
+1	row number
+2	top left column of current window
+3	top left row of current window
+4	bottom right column of current window
+5	bottom right row of current window
+6	current attribute for current window
+7	?scroll - 1 = scroll, 0 = no scroll
+8	video page number of current window
+9	temporary byte location
+10	save address of current window
+12	video board base address of current window

CURRENT FO,ST,UV,0,UN,(-> addr)
Leaves the address of a variable specifying the vocabulary into which new word definitions are to be entered. The CURRENT vocabulary is searched when the search of the CONTEXT vocabulary ends.

CURSOR FO,NO,UV,0,UN,(-> addr) (TRS-80)
Leaves the address of a user variable where the cursor character is stored. As delivered, it contains 176 decimal. See Appendix A12.
It can easily be changed. Try DECIMAL 36 CURSOR !
(Model III users try 23 EMIT 196 CURSOR !)

CVARIABLE FO,NO,,17/22,UN,"c-variable"
A defining word used in the form:

```
CVARIABLE <name>
```

to create a dictionary entry for <name> and allot one byte for storage in the parameter field. (MMSFORTH initializes it to zero.) The application must initialize the stored value for 79-STANDARD. When <name> is later executed, the parameter field address is placed on the stack.

CVBLK CA,NO,,58/77,UN,"c-v-block",(addr n ->)
Verifies Block n on tape by reading it in and comparing it to the data at addr. The data to be compared must already be at addr. Most often used after a write.
Returns _Error ok_ or _ok_.

D#IN DP,NO,,44/63,UN,"d-number-in",(-> d)
Inputs a double-precision signed integer number (contains a decimal point) and puts it on the stack. Sets HI# and #PT. Included blank characters or other bad input returns _? Redo ?_.

D* DP,NO,,48/67,NE,"d-times",(d1 d2 -> d3)
Leaves the double number product of d1 times d2.

D*/ DP,NO,,48/67,NE,"d-times-divide",(d1 d2 d3 -> d4)
Multiplies d1 by d2, then divides by d3. n4 is rounded toward zero. The product of d1 times d2 is maintained as a 64-bit value for greater precision.

D*/MOD DP,NO,,48/67,NE,"d-times-divide-mod",(d1 d2 d3 -> d4 d5)
Multiply d1 by d2, divide the 64-bit result by d3 and leave the remainder d4 and the quotient d5. The remainder has the same sign as d1.

D*S DP,NO,,45/64,UN,"d-times-s",(ud un -> ut)
Multiplies unsigned double-precision by single-precision giving triple-precision result.

D+ FO,ST,,0,CH,"d-plus",(d1 d2 -> d3)
Leaves the arithmetic sum of d1 plus d2.

D- FO,EX,,34/40,CH,"d-minus",(d1 d2 -> d3)
Leaves the difference of d1 minus d2.

D/ DP,NO,,47/66,CH,"d-divide",(d1 d2 -> d3)
Divides d1 by d2 leaving quotient d3. d3 is rounded toward zero.

D/MOD DP,NO,,47/66,NE,"d-divide-mod",(d1 d2 -> d3 d4)
Divides d1 by d2 leaving remainder d3 and quotient d4. d3 has the same sign as d1.

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

D. DP,EX,,44/63,CH,"d-dot",(d ->)
Outputs d as an integer, converted according to BASE followed by one blank. Only a negative sign is displayed.

D.R DP,EX,,44/63,CH,"d-dot-r",(d n ->)
Outputs d as an integer, converted according to BASE, right aligned in an n character field. Display the sign only if negative.

D0= FO,EX,,0,CH,"d-zero-equals", (d -> flag)
True if d is zero.

D< DP,EX,,43/62,CH,"d-less-than",(d1 d2 -> flag)
True if d1 is less than d2.

D= DP,EX,,44/63,CH,"d-equal",(d1 d2 -> flag)
True if d1 equals d2.

D? GR,NO,,54,UN,"d-question",(n1 n2 -> flag) (TRS-80)
In a double-width graphics grid consisting of 48 rows and 64 columns of "square" graphic points, returns true if the point at Row n1, Column n2 is set.

```
5 50 D?
```

checks the condition of the 51st point in Row 6 (where the upper left screen point is 0,0).

D?# FO,NO,,0,NE,"d-question-number", (IBM PC)
(n -> n1 n2 n3 n4)
Given a block number returns the drive n1, track n2, start sector n3, and the number of sectors n4 for that block.

DABS DP,EX,,43/62,CH,"d-abs",(d1 -> d2)
Leave the positive double number d2 as the absolute value of d1. Valid range is 0 to 2,147,483,647.

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

DARRAY AR,NO,,49/68,UN,"d-array",(n ->)
A defining word used in the form:
n DARRAY <name>
to create a dictionary entry for <name>, consisting of n+1 four-byte elements (indexes 0 to n).
When <name> is later used it is in the form:
n <name>
where the address of Element n is put on the stack.
Define an array named B with 16 elements.
15 DARRAY B
To set Element 5 equal to Element 7:
7 B 2@ 5 B 2!

DATE CL,NO,,60/79,NE
Outputs system date in the format MM/DD/YY. (See SET-DATE.)

DCLR GR,NO,,54,UN,"d-clear",(N1 N2 ->) (TRS-80)
In a double-width graphics grid consisting of 48 rows of 64 "square" graphic points, clears the point at Row n1, Column n2.

DDEN FO,NO,,23,NE,"d-den",(n ->) (TRS-80 M.3 Disk only)
Sets Drive n to double-density. See SDEN .

DECIMAL FO,ST,,16/21,UN
Sets input-output number conversion BASE to 10.

DEFINITIONS FO,ST,,17/22,UN
Sets CURRENT to the CONTEXT vocabulary so that subsequent definitions will be created in the vocabulary previously selected as CONTEXT.

DEPTH FO,ST,,32/47,NE,(-> n)
Leaves the number of 16-bit values contained in the user stack before n was added.

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

DIR FO,NO,,25/38,CH,"d-i-r"
Loads the block pointed to by DIRBLK in order to display the pseudo-directory.

DIRBLK FO,NO,,15/20,NE,"dir-block",(-> n)
Leaves block number n of DIRectory block.

DISK CA,NO,,58/77,UN (TRS-80 Disk system only)
Directs mass storage read/write operations (RBLK, WBLK, etc.) to disk drive(s).
See TAPE .

DISKDATA FO,NO,UV,0,NE,(n -> addr) (Disk System only)
Array containing information about the physical characteristics of the disk drives. See Appendix A12.

```
0 DISKDATA 74 DUMP
```

DISK-TAPE CA,NO,,58/77,UN,(n1 n2 ->) (Disk System only)
Transfers consecutive disk blocks to tape starting at Block n1 for a count of n2.
(Prepare recorder in advance, by positioning and pressing Play and Record.)

DISP-B# FO,NO,,52,NE,"display-b-number",(block# ->) (IBM PC)
Displays block number in decimal in absolute and drive number, relative block number formats.

DMAX DP,EX,,43/63,NE,"d-max",(d1 d2 -> d3)
Leaves the larger of two signed double numbers.

DMIN DP,EX,,43/63,CH,"d-min",(d1 d2 -> d3)
Leaves the smaller of two signed double numbers.

DNEGATE FO,EX,,0,CH,"d-negate",(d -> -d)
Leaves the double number two's complement of a double number; i.e., the difference, 0 less d.

DO FO,ST,IM&CO,22/39,UN,(n1 n2 ->)
Use in a colon-definition in the format:
DO ... LOOP or
DO ... +LOOP
Begins a finite loop which will terminate based on control paramenters. The loop index begins at n2, and terminates based on the limit n1. At LOOP the index is incremented by +1; at +LOOP by the positive or negative value currently on the stack. The range of a DO ... LOOP is determined by the terminating word. DO ... LOOPs always execute at least once and may be nested. In MMSFORTH, the indexes are moved to the return stack during execution.

DOES> FO,ST,IM&CO,0,UN,"does"
Defines the run-time action of a word created by a high-level defining word.
Used in the form:
: <name> ... CREATE ... DOES> ... ;
and then <name> <namex>
Marks the termination of the creation part of the defining word <name> and begins the definition of the run time action for words that will later be defined by <name>. On execution of <namex> the sequence of words between DOES> and ; will be executed, with the address of <namex>'s parameter field on the stack.

DP FO,NO,UV,0,CH,"d-p",(-> addr)
Dictionary Pointer leaves the address of the user variable containing the address of the top of the dictionary.

DRDSECS FO,NO,,0,UN,"d-read-secs", (Disk System only)
(addr n1 n2 n3 n4 -> flag)
Reads any number of consecutive disk sectors into memory location at addr from Drive n1, Track n2, and Sector n3 for sector-count n4 leaving zero if no error was detected.
Maximum numbers are: drives=8, tracks=255, sectors=255, number of sectors=255.
ALL SECTORS READ MUST BE ON THE SAME DRIVE.

DROP FO,ST,,0,UN,(n ->)
Drops the top number from the stack.

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

DSET GR,NO,,54,UN,"d-set",(n1 n2 ->) (TRS-80)
In a double-width graphics grid consisting of 48 rows of 64 "square" graphic points, sets a double-width graphics element at Row n1, Column n2.

DU* DP,NO,,48/67,UN,"d-u-times", (ud1 ud2 -> uq)
Multiplies unsigned double-precision numbers ud1 and ud2 giving quadruple number (64-bit) unsigned result.

DU/MOD DP,NO,,47/66,NE,"d-u-divide-mod",(uq ud1 -> ud2 ud3)
Performs unsigned division of quadruple number (64-bit) uq by ud1, leaving the remainder ud2 and quotient ud3. All values are unsigned.

DU< DP,EX,,44/63,NE,"d-u-less-than",(ud1 ud2 -> flag)
True if ud1 is less than ud2. Both numbers are unsigned.

DUMP FO,NO,,35/53,UN,(addr n ->)
Outputs the values contained in memory starting at addr for n bytes.

```
HEX 8000 20 DUMP
```

dumps 32 bytes starting at address 8000 Hex. (On IBM PC, see also LDUMP.)

DUP FO,ST,,0,UN,"dupe",(n -> n n)
Leaves a duplicate copy of the top stack number.

Dup-name: Special system comment, indicating compilation of a name with a similar one already in the Dictionary. Comment may be suppressed with: 0 21 MMS 25 + C! .

DWTSECS FO,NO,,0,UN,"d-write-secs" (Disk System only)
(addr n1 n2 n3 n4 -> flag)
Writes any number of consecutive disk sectors from memory location at addr from Drive n1, Track n2, and Sector n3 for sector-count n4 leaving zero if no error was detected. Maximum numbers are: drives=8, tracks=255, sectors=255, number of sectors=255.
ALL SECTORS WRITTEN MUST BE ON THE SAME DRIVE.
WARNING: This word ignores PBLK, the software block protect feature!

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

E FO,NO,,39/58,CH
EDITs the screen specified in SCR.

E? GR,NO,,54/73,UN,"e-question",(n1 n2 -> flag)
In a graphics grid (48 rows by 128 columns on TRS-80, 50 rows by 80 columns on IBM PC), returns true if the point at Row n1, Column n2 is set.

```
5 50 E?
```

checks the condition of the 51st point in Row 6 (where top is the upper left screen point is 0,0).

EBLK FO,NO,UV,0,NE,"e-block",(-> addr)
Leaves address of user variable which holds the (16-bit) block number in error message. (See EPOS.)

ECLR GR,NO,,54/73,UN,"e-clear",(n1 n2 ->)
In a graphics grid (48 rows by 128 colums on TRS-80, 50 rows by 80 columns on IBM PC), clears a graphics element at Row n1, Column n2. (The upper left screen point is 0,0.)

EDIT FO,NO,,39/58,CH,(n ->)
Sets SCR to n and displays Screen n in EDITOR mode.

EDITOR FO,NO,IM,37/55,UN
The name of the Editor vocabulary. When this name is executed, EDITOR is established as the CONTEXT vocabulary.

EEDIT FO,NO,,39/58,NE,"e-edit"
EDITs block specified in error message (EBLK), positioning cursor at 1st character of the error word (EPOS).

ELSE FO,ST,IM&CO,16/21,UN
Used in a colon-definition in the form:

```
IF ... ELSE ... THEN
```

ELSE executes after the true part following IF. ELSE forces execution to skip to just after THEN. It has no effect on the stack.
The words that follow ELSE are executed if top-of-stack was zero (false) when IF was invoked.

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

EMIT FO,ST,,0,CH,(char ->)
Outputs char. Can be used to print special characters.
36 EMIT
will print a "$" (36 decimal is the ASCII code for $).

EMPTY-BUFFERS FO,ST,,32/47,CH
Mark all buffers as empty. UPDATEd blocks are not written to mass storage. The actual contents of the buffers are not changed.

ENCODE FO,NO,,0,NE,,(addr -> d) (IBM PC)
Encodes string at addr into 32-bit hashed representation.

ENTER FO,NO,,21/38,UN,
Outputs (Enter) and pauses until the Enter key is pressed.

EPOS FO,NO,UV,0,NE,"e-position",(-> addr)
Leaves address of user variable holding error position offset (number of 16-bit characters) from upper left in block of error message. (See EBLK.)

ERASE FO,NO,,26/40,UN,(addr n ->)
Stores binary zeroes in memory for n bytes starting at addr.

ESET GR,NO,,54/73,UN,"e-set",(n1 n2 ->)
In a graphics grid (48 rows by 128 colums on TRS-80, 50 rows by 80 columns on IBM PC), sets a graphics element at Row n1, Column n2. (The upper left screen point is 0,0.)

EXECUTE FO,ST,,0,UN,(addr ->)
Executes the dictionary entry whose compilation address is on the stack.

EXIT FO,ST,,0,NE
When compiled within a colon-definition, terminates execution of that definition at that point. (May not be used within a DO ... LOOP.) When executed from a load screen, terminates interpretation of the screen at that point.

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

EXPECT FO,ST,,0,UN,(addr n ->)
Inputs characters to memory beginning at addr, upward, until carriage-return or the count of n has been received. Two nulls are added at the end of text. MMSFORTH's Input Line Editor redefines the word (see Appendix A12 and Option Select Block, typ. Block 15/20).

FILL FO,ST,,26/40,CH,(addr n char ->)
Fills a range of memory with n copies of char .

```
15360 1024 191 FILL QUIT
```

fills TRS-80 video memory with all-white graphics characters.

FIND FO,ST,,0,NE,(-> addr)
Use in the form:

```
FIND <name>
```

Leaves the compilation address of the next word <name> to be accepted from the input stream. If that word cannot be found in the dictionary after a search of CONTEXT and CURRENT, leaves zero.

FLUSH FO,NO,,32/47,UN
Forces all updated blocks to be written to disk. A synonym for SAVE-BUFFERS.

FORGET FO,ST,,17/22,UN
Execute in the form:

```
FORGET <name>
```

Deletes from the dictionary all words added to the dictionary after <name> including the word <name>, regardless of their vocabulary. Cannot be used within a colon-definition.

FORTH FO,ST,IM,0,UN
The name of the primary vocabulary. Execution makes FORTH the CONTEXT vocabulary. New definitions become a part of FORTH until a differing CURRENT vocabulary is established. User vocabularies conclude by 'chaining' to FORTH, so it should be considered that FORTH is 'contained' within each user's vocabulary.
It is IMMEDIATE, so it may be invoked inside a definition.

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

FULL-CUR FO,NO,,38,NE,"full-cursor",(->) (IBM PC)
Sets full height cursor.

GET-CHR FO,NO,,0,NE,"get-character",(-> byte) (IBM PC)
Returns the character at the cursor on the video screen.

GET-CHRS FO,NO,,40,NE,(adr count ->) (IBM PC)
Like MOVE-CHRS-FROM-SCRN, but turns off video for color board and returns cursor to position before command was executed.

GET-DATE CL,NO,,60(??)/79,NE,(-> n1 n2 n3)
Returns n1 Month, n2 Day, n3 Year

GTC FO,NO,,22,NE,"get-cursor",(-> row column) (IBM PC)
Returns row and column position of cursor.

H/L CA,NO,,56,NE,"h-slash-l" (TRS-80 M.3 only)
Calls ROM routine to set cassette speed. Then respond with H for high-speed, or L for low-speed.

HERE FO,ST,,0,UN,(-> addr)
Leaves the address of the next available byte at the top of the dictionary.

HEX FO,NO,,16/21,UN,"hex"
Sets I/O number conversion BASE to 16 (hexadecimal).

HI# FO,NO,UV,0,UN,"high-number",(-> addr)
Leaves address of user variable which holds the high word (16 bits) of the last number input. All numbers are input to 32-bit precision, but if the number input did not have a decimal point only the low word (16 bits) is pushed onto the stack. The value of the high word (16 bits) is put into HI#.

HOLD FO,ST,,0,UN,(char ->)
Inserts char into a pictured numeric output string. May only be used between <# and #> .

HSYNC FO,NO,,48,NE,"h-sink",(n ->) (IBM PC)
Horizontal synchronization; moves display on screen right for positive n and left for negative n.

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

I FO,ST,,21/39,UN,(-> n)
For 79-STANDARD, copies the loop index onto the data stack. May only be used in the form:
DO ... I ... LOOP or
DO ... I ... +LOOP
In MMSFORTH, copies the top of the return stack onto the user stack; does not alter the return stack. May be used to get the loop index or in the same way R@ is used.

I' FO,NO,CO,21/39,NE,"i-prime",(-> n)
For 79-STANDARD, used only within a colon-definition executed only from within a DO ... LOOP to return the corresponding loop index.
For MMSFORTH, returns the second on the return stack. If used within a loop, it gives the loop limit. If used within a colon-definition which was called from within a loop, it gives the loop index.

IBM FO,NO,,48,NE,(drive# ->) (IBM PC)
Sets drive# to read and write IBM formatted disks. See M.3 .

IF FO,ST,IM,16/21,UN,(flag ->)
Used in a colon-definition in the form:
flag IF ... ELSE ... THEN or
flag IF ... THEN
If flag is true, the words following IF are executed and the words following ELSE to THEN are skipped. The ELSE part is optional.
If flag is false, the words between IF and ELSE, or between IF and THEN (when no ELSE is used), are skipped.
IF ... ELSE ... THEN conditionals may be nested.

IMMEDIATE FO,ST,,16/21,UN
Marks the most recently defined dictionary entry as a word which will be executed when encountered during compilation rather than compiled. Compilation of an IMMEDIATE word may be forced with the word [COMPILE].

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

IN$ ST,NO,,50/69,UN,"in-string",(-> $)
Inputs a string into PAD, and puts PAD's present address on the stack. A ? is output to prompt for input.

```
20 $VARIABLE XX
: REPEATS  IN$ XX $!  CR 0 DO XX $. SPACE LOOP ;
50 REPEATS   Enter  ?
```

(try "MMSFORTH!", "I Love You!", "Aligned", etc.)

INDEX FO,NO,,35/53,CH,(n1 n2 ->)
Starting at Screen n1 for a count of n2, output Line 0 of each screen containing printable text in Line 0.
By convention Line 0 contains a title.

```
55 67 :R INDEX
```

Lists Screens 55 through 67. :R converts the start and end screens to start and count.

INKEY$ ST,NO,,52/71,UN,"in-key-string",(-> $)
Simulates INKEY$ function of Level II BASIC. Checks to see if an input character is available. If so, creates a string of the character in PAD; if not, creates a length zero string.

INP FO,NO,,34/49,NE,"in-p",(n -> byte)
Inputs a byte from Port n.

INSTR ST,NO,,51/70,UN,"in-s-t-r",($1 $2 -> n)
Searches $1 to see if it contains $2. Returns relative starting position or 0.

J FO,ST,CO,21/39,UN,(-> n)
For 79-STANDARD, copies the index of the next outer loop onto the user stack. May only be used within a nested DO ... LOOP in the form:

```
DO ... DO ... J ... LOOP ... LOOP
```

In MMSFORTH, copies the third item on the return stack onto the user stack. Usually used to get the index of the next outer loop in a nested DO ... LOOP as described for 79-STANDARD above.

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

KEY FO,ST,,0,UN,(-> char)
Leaves the ASCII value of the next available character from the current input device. If the current input device is the keyboard, blinks the cursor while waiting for a key to be pressed.

```
KEY . $ 36
```

(36 is the decimal ASCII code for $).

L FO,NO,35/53,CH
LISTs the screen specified in the user variable SCR.

L! LA,NO,,83,NE,"l-store",(n daddr ->) (IBM PC)
Long-address version of ! .

L2! LA,NO,,83,NE,"l-two-store",(d daddr ->) (IBM PC)
Long-address version of 2! .

L2@ LA,NO,,83,NE,"l-two-fetch",(daddr -> d) (IBM PC)
Long-address version of 2@ .

L<CMOVE LA,NO,,84,NE,"l-reverse-c-move",(daddr1 daddr2 n ->) (IBM PC)
Long-address version of <CMOVE.

L@ LA,NO,,83,NE,"l-fetch",(daddr -> n) (IBM PC)
Long-address version of @ .

LABEL FO,NO,,20/37,UN
Creates a header in the form of a variable, and sets CONTEXT to the ASSEMBLER vocabulary.
This word is normally used to define Assembly language subroutines completed by RET instead of NEXT. They are not the same as code procedures. When a word defined using LABEL is executed it puts its parameter field address on the stack for the following CALL or JMP instruction.

LAST FO,NO,UV,0,NE,(-> addr)
A variable containing the address of the beginning of the last dictionary entry made (which might not be a complete or valid entry).

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

LC! LA,NO,,83,NE,"l-c-store",(n daddr ->) (IBM PC)
Long-address version of C! .

LC@ LA,NO,,83,NE,"l-c-fetch",(daddr -> byte) (IBM PC)
Long-address version of C@ .

LCMOVE LA,NO,,84,NE,"l-c-move",(daddr1 daddr2 n ->) (IBM PC)
Long-address version of CMOVE .

LDUMP LA,NO,,85,NE,"l-dump",(daddr n ->) (IBM PC)
Long-address version of DUMP .

LEAVE FO,ST,CO,22/39,UN
Sets the limit of a DO ... LOOP equal to the current value of the index so that a loop will be terminated prematurely. The index itself remains unchanged, and execution proceeds normally until LOOP or +LOOP is encountered.

LEFT$ ST,NO,,50/69,UN,"left-string",($1 n -> $2)
Starting from the left, take n characters from $1 to make $2. $2 is placed in PAD.

LEN ST,NO,,52/71,UN,($ -> n)
Returns the character length of $.

LFILL LA,NO,,84,NE,"l-fill", (daddr n char ->) (IBM PC)
Long-address equivalent of FILL, using word-count and 8-bit char.

LIST FO,ST,,35/53,UN,(n ->)
Lists the ASCII symbolic contents of Screen n with line numbers onto current output device, sets SCR to n.

LITERAL FO,ST,IM,16/21,NE,(n ->)
If compiling, then compile the stack value n as a 16-bit literal which, when later executed, will leave n on the stack.

LOAD FO,ST,,0,UN,(n ->)
Begins interpretation of source text in Screen n by making it the input stream (from >IN and BLK). If interpretation is not terminated explicitly it will be terminated when the input stream is exhausted. Control then returns to the input stream containing LOAD, determined by the input stream locators >IN and BLK.

LOADS FO,NO,,0,UN,(n1 n2 ->)
Loads multiple blocks starting with Screen n1 for n2 blocks, all via one block buffer (is overridden by -->).

LOOP FO,ST,IM&CO,22/39,CH
Increment the DO ... LOOP index by one, terminating the loop if the new index is equal to or greater than the limit. The limit and index may be in the range -32,768 to 32,767.

LWFILL LA,NO,,84,NE,"l-w-fill", (daddr n1 n2 ->) (IBM PC)
Long-address equivalent of FILL, using n1 word-count and n2 word-value (i.e., 16-bit values). See PAINT example on System Disk.

M* FO,NO,,33/51,UN,"m-times",(n1 n2 -> d)
Multiplies n1 by n2 leaving d.

M*/ DP,NO,45/64,UN,"m-times-divide",(d1 n1 n2 -> d2)
Multiplies d1 by n1 then divides by n2 giving d2. The intermediate result is maintained as a 48-bit (triple) number.

M+ FO,NO,,0,UN,"m-plus",(d1 n -> d2)
Adds n to d1 giving d2, (mixed precision).

M- FO,NO,,0/51,UN,"m-minus",(d1 n -> d2)
Subtracts n1 from d1 giving d2 (mixed precision).

M.3 FO,NO,,48,NE,"m-dot-three",(drive# ->) (IBM PC)
Sets drive# to read and write TRS-80 Model III formatted disks. See IBM.

M/ FO,NO,,33/51,UN,"m-divide",(d n1 -> n2)
Divides d by n1 giving n2.

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

M/MOD FO,NO,33/51,UN,"m-divide-mod",(d n1 -> n2 n3)
Divides d by n1 giving remainder n2 and quotient n3.

MARGIN FO,NO,UV,29&30/44&45,NE,(-> addr)
Leaves address of user variable containing current margin indent for printer. MARGIN 2+ C@ is the current character position on the line.

MAX FO,ST,26/0,UN,"max",(n1 n2 -> n3)
Leaves the greater of two numbers.

MID$ ST,NO,,50/69,UN,"mid-string",($1 n1 n2 -> $2)
Returns a substring of length n2 from $1 starting n1 characters into $1.

MIN FO,ST,,26/0,UN,"min",(n1 n2 -> n3)
Leaves the lesser of two numbers.

MMS FO,NO,UV,0,NE
Array which contains addresses of various system routines. See Appendix A12.

MOD FO,ST,,33/51,UN,"mod",(n1 n2 -> n3)
Divides n1 by n2, leaving the remainder n3, with the same sign as n1.

MODULUS RA,NO,,53/72,UN,(-> n)
Constant containing modulus used in Lehmer multiplication congruential random number generator.

MOVE FO,ST,,36/54,CH,(addr1 addr2 n ->)
Moves n 16-bit memory cells from addr1 into memory at addr2. CMOVE is preferred for 8-bit machines, such as the TRS-80.

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

MOVE-CHRS-TO-SCRN FO,NO,,0,NE,,(addr count ->) (IBM PC)
Moves count characters to screen at current cursor position from addr.

MOVE-CHRS-FROM-SCRN FO,NO,,0,NE,,(addr count ->) (IBM PC)
Moves count characters from screen at current cursor position to addr.

MUL RA,NO,,53/72,UN
Constant containing multiplier used in Lehmer multiplication congruential random number generator.

MYSELF FO,NO,IN&CO,16/20,NE
Causes address of word currently being compiled to be compiled into itself. Allows recursive programming in 79-STANDARD.

NCASE FO,NO,IM&CO,25/20,UN,"n-case",(n ->)
Begins numeric case structure in the form:

```
NCASE n1 n2 n3 " <name> <name> <name>
    OTHERWISE ... CASEND
```

At compilation, it is followed by one blank and a list of one-byte values each of which matches an appropriate routine in the dictionary. The list of values is terminated with a " followed by the matching list of routines. When executed compares n with each defined character and executes appropriate <name>.

```
NCASE 1 13 223 " 1RTN 13RTN 223RTN
    OTHERWISE ." BAD" CR CASEND
```

If top item on stack = 1 then 1RTN is executed.
If top item on stack is not equal to any of the defined characters, control passes to code following OTHERWISE (if present) or CASEND.

NEGATE FO,ST,,0,CH,(n -> -n)
Leaves the two's complement of a number; i.e., the result of 0 minus n.

NO-CUR FO,NO,,38,NE,"no-cursor",(->) (IBM PC)
Turns cursor off.

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

NOT FO,ST,,26/40,UN,(flag1 -> flag2)
Reverses the truth value of flag1. 0 becomes 1, and all other values become 0. (Identical to 0= .)

NUMBER FO,NO,,0,UN,(addr1 -> n addr2) or (addr1 -> d addr2)
Converts the numeric ASCII string at addr1-1 to binary according to the current value of BASE. Leaves n or d (d if a decimal point was found in the number) and sets HI# and #PT. addr2 points to the first non-numeric character.

OCTAL FO,NO,,16/21,UN
Sets number conversion base at BASE to 8.

OR FO,ST,,26/0,UN,(n1 n2 -> n3)
Leaves the bitwise inclusive-OR of two numbers.

OTHERWISE FO,NO,IM,25/21,UN
Optional default condition for ACASE and NCASE; if used, can be followed by any Forth word(s).
Used in the form:
ACASE ..." ... OTHERWISE ... CASEND or
NCASE ..." ... OTHERWISE ... CASEND

OUTP FO,NO,,34/49,NE,"out-p",(byte n ->)
Output byte to port n.

OUT/WORD FO,NO,,40,NE,"out-slash-word",(cfa ->) (IBM PC)
Defining word for output switching words like CRT, PRINT.

OVER FO,ST,,0,UN,(n1 n2 -> n1 n2 n1)
Leaves a copy of the second number on the stack.

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

P! FO,NO,,49,NE,"p-store",(n port# ->) (IBM PC)
Outputs n to port# and port#+1.

P-IT FO,NO,,44,NE,"print-it",(char ->) (IBM PC)
Output char (character) to printer.

P@ FO,NO,,49,NE,"p-fetch",(port# -> n) (IBM PC)
Inputs 16 bit word from port# and port#+1.

P&C FO,NO,,47,NE,"p-and-c"(char ->) (IBM PC)
Prints character on video and printer.

PAD FO,ST,,0,UN,(-> addr)
The starting address of a scratch area used to hold character strings for intermediate processing. PAD moves as definitions are added to and deleted from the dictionary. In MMSFORTH, PAD is located 65 bytes above HERE.

PAGE FO,ST,,21/38,CH
Clear the terminal screen or perform an action suitable to the output device currently active (Top-of-Form on printer, etc.)

PBLK FO,NO,,0,UN,"p-block",(-> addr) (Disk System only)
Leaves address of a user variable which holds the lower limit of the non-software-write-protected disk blocks. MMSFORTH's "rubber write-protect tab", PBLK allows a protected system area and an unprotected data area on a single diskette. The user sets PBLK to a block number below which the normal virtual mass storage feature will not write, but will give an error message.
WARNING: DWTSECS overrides this protection feature!

PC! FO,NO,,38,NE,"p-c-store", (byte port# ->) (IBM PC)
Outputs byte to port number.

PC@ FO,NO,,490,NE,"p-c-fetch",(port# -> byte) (IBM PC)
Inputs byte from port#.

PCRT FO,NO,,32/47,UN,"p-c-r-t",(->)
Sets current output device to both video screen and printer.

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

PICK FO,ST,,34/49,NE,(n1 -> n2)
Return the contents of the n1-th stack value, not counting n1 itself.
2 PICK is equivalent to OVER.

PINIT FO,NO,,44,NE,"printer-init",(->) (IBM PC)
Reinitialize parallel printer driver.

PLIST FO,NO,,35/53,UN,"p-list",(n ->)
Does BLIST of Screen n with extra CR's (line feeds).

PLISTS FO,NO,,35/53,UN,"p-lists",(n1 n2 ->)
Does multiple PLISTs starting at Screen n1 for a count of n2. Three screens exactly fill an 11-inch page.

PRINT FO,NO,,29&30/47,UN,(->)
Sets the printer to be the current output device.
On the TRS-80 this is normally the parallel port via the Level II or Model III ROM printer-driver. References the DCB address, so it may be linked with a preloaded serial printer-driver routine. To do so: preset MMSFORTH System's RAM size appropriately; load driver; then boot the re-sized MMSFORTH System and store the routine's entry address at 16422.
On the IBM PC normally uses the parallel printer driver. A serial printer driver is supplied and can be activated by changing Block 20.
See the alternate printer-drivers for examples of how a custom printer driver might be done in Forth. For safety, make your routine save and restore all registers between character I/O.

PTC FO,NO,,26/0,UN,"put-cursor",(n1 n2 ->)
Puts cursor at video screen line n1 and column n2.

```
: HELLO PAGE 8 20 PTC
    ." Hi there, programmer!" CR ;
```

PUT-CHRS FO,NO,,40,NE,"put-characters",(adr count ->) (IBM PC)
Like MOVE-CHRS-TO-SCRN, but turns off video for color board and returns cursor to position before command was executed.

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

QUERY FO,ST,,0,NE
Accept input of up to 80 characters or until a CR from the operator's terminal, into the terminal input buffer. WORD may be used to accept text from this buffer as the input stream, by setting >IN and BLK to zero.

QUESTION FO,NO,,0,UN
Repeats the last word from the input stream executed by the text interpreter, issues a question-mark, then empties both stacks and returns control to the operator. No ok is issued. Normally used for reporting errors. If the input stream is from blocks, it also types the block, line and column numbers at which the error occurred and sets EBLK to the error block number and EPOS to the position within the block at which the error occurred, for use with EEDIT .

QUIT FO,ST,,0,CH
Clears the return stack, sets execution mode, and returns control to the terminal. No ok is issued.

R> FO,ST,,0,UN,"r-from",(-> n)
Transfers n from the return stack to the user stack.
In 79-STANDARD, may only be used in a colon-definition.
In MMSFORTH, may be used anywhere.

R@ FO,ST,,0,NE,"r-fetch",(-> n)
Copy the number on the top of the return stack to the user stack.
In 79-STANDARD, may only be used in a colon-definition.
In MMSFORTH it is identical to I and may be used anywhere.

RANDOMIZE RA,NO,53/72,UN
Randomizes the random number generator. Performs the same function as Level II and Model III BASIC's RANDOM command.

RBLK FO,NO,0,UN,"r-block",(addr n ->)
Virtual read block routine.
Reads Screen n into memory at addr. Uses routine whose code field address (CFA) is stored at 21 MMS 6 + (See Appendix A12 for a listing of the MMS Table.)

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

REPEAT FO,ST,IM&CO,16/21,CH
Used in a colon-definition in the form:
BEGIN ... WHILE ... REPEAT
At run-time, if WHILE test was true REPEAT returns to just after the corresponding BEGIN. (See WHILE).

RFR RA,NO,,53/72,UN,"r-f-r",(-> n)
Puts the value of the refresh register on the stack.

RIGHT$ ST,NO,,50/69,UN,"right-string",($1 n -> $2)
Starting from the right, takes n characters from $1 to make $2. $2 is placed in PAD.

RN1 RA,NO,,53/72,UN,"r-n-one"
Generates the next random number and stores it in SEED.

RND RA,NO,,53/72,UN,"r-n-d",(n1 -> n2)
Generates a random number n2 between 1 and n1 inclusive.

ROLL FO,ST,,34/49,NE,(... nb-> ...)
Extract the n-th stack value to the top of the stack, not counting n itself, moving the remaining values into the vacated position. n must be greater than zero.
3 ROLL is equivalent to ROT

ROT FO,ST,,0,UN,"rote",(n1 n2 n3 -> n2 n3 n1)
Rotates the top three values, bringing the deepest to the top.

RP AS,NO,UV,20,CH,"r-p",(-> addr) (TRS-80)
Leaves address of Return-stack Pointer.

S0 FO,NO,UV,0,NE,"s-zero",(-> addr)
Leaves the address of a user variable containing the address of the bottom of the user stack. Note that S0 is a variable, while 'S is a constant.

S>L LA,NO,,83,NE,"s-to-l",(addr -> daddr)
Converts a short-address (relative to start of FORTH) to its long-address (absolute, double-precision number) equivalent.

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

SAVE-BUFFERS FO,ST,,36/54,NE
Write to mass storage, all the buffers that have been flagged as UPDATEd.

SCAN-CODE F0,NO,,0,NE,,(-> addr) (IBM PC)
Returns address of byte containing scan code of last chraracter input by ?KEY or KEY.

SCR FO,ST,UV,35/53,UN,"s-c-r",(-> addr)
Leaves the address of a user variable containing the number of the screen most recently listed or edited.

SDEN FO,NO,,23,UN,"s-den",(n ->) (TRS-80 M.3 Disk System Only)
Sets Drive n to single-density. See DDEN.

SEED RA,NO,UV,53/72,UN,(-> addr)
Variable, initially containing seed used in Lehmer multiplication congruential random number generator.

SET-BORDER FO,NO,,48,NE,(color ->) (IBM PC)
Sets border color of video display.

SET-COLOR FO,NO,,48,NE,(background foreground ->) (IBM PC)
Set color and monochrome display attributes in current window. Background and foreground may be from 0 to 15. Backgrounds of 8 or greater create blinking characters. (See COLOR in IBM BASIC Manual, or see IBM Technical Reference.)

SET-DATE CL,NO,,60/79,NE,(n1 n2 n3 ->)
Sets the system date to n1 month, n2 day, n3 year.
Example:

```
12 25 83 SET-DATE
```

SET-MODE FO,NO,,40,NE,(char ->) (IBM PC)
Sets video display mode to mode# represented by char. (See IBM Technical Reference, p. 2-58, "Mode Select Register".)

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

SET-TIME CL,NO,,60/79,NE,(n1 n2 n3 ->)
Sets the system time to n1 hours, n2 minutes, n3 seconds.
Example:

```
23 45 0 SET-TIME
```

SET-WINDOW FO,NO,,48,NE,(flag n1 n2 n3 n4 n5 ->) (IBM PC)
Defining word for windows on video screen, where:
flag: 0 = no scroll; 1 = scroll
n1: attribute; first 4 bits are background color or monochrome attribute; last 4 bits are foreground color or monochrome attribute. In Hex, shows as two characters. Background: 0-7 are eight colors; 8-15 are blinking color. Foreground: 0-15 are 16 colors.
n2: bottom-right-row; valid values are 0-24. Note: if this or any of the next three values is too large or overlap, it may make your window go off the bottom of the video display or be invisible.
n3: bottom-right-column; valid values are 0-79.
n4: top-left-row; valid values are 0-24.
n5: top-left-column; valid values are 0-79.
W/0 is the standard full-screen, white-on-black window.
Example in color:

```
HEX 1 53 DECIMAL 20 30 0 0
   SET-WINDOW W/CYAN-ON-MAGENTA
```

sets a scrolling window which is 30 wide and 20 long (from 0,0 to 20,30) with characters of Color 5 (Cyan) on a background of Color 3 (Magenta). It can be invoked by the word, W/CYAN-ON-MAGENTA . Changing the 53 to D3 will cause this window to blink when invoked.

SIGN FO,ST,CO,0,CH,(n ->)
Inserts an ASCII "-" (minus sign) into the pictured numeric output string if n is negative.

SPACE FO,ST,,0,UN
Outputs a single ASCII blank character (32 decimal).

SPACES FO,ST,,21/38,UN,(n ->)
Outputs n spaces. Will output no spaces if n is less than or equal to zero.

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

STATE FO,ST,UV,0,UN,(-> addr)
Leaves the address of a one-byte user variable containing the compilation state. A non-zero content indicates compilation is occurring.

```
STATE C?
```

prints 1 in compile mode or 0 in execute mode.

STR$ ST,NO,,52/71,UN,"s-t-r-string",(n -> $)
Converts n to a string in PAD and leaves PAD address on stack.

STRING$ ST,NO,,52/71,UN,"string-string",(n char -> $)
Returns a string of characters defined by char of a length of n.

```
10 $" *" STRING$ $.                    or
$CONSTANT STARS *"  10 STARS STRING$ $.
```

Both return **********.

SWAP FO,ST,,0,UN,(n1 n2 -> n2 n1)
Exchanges the top two items on the stack.

T/S DP,ST,,45/64,UN,"t-divide-s",(ut un -> ud)
Divides a triple number by a single number leaving a double number result.

TAPE CA,NO,,58/77,UN (Disk system only.)
Directs mass storage read/write operations (RBLK, WBLK, etc.) to the cassette recorder.

TAPE-DISK CA,NO,,58/77,UN,(n1 n2 ->) (Disk System only)
Transfers consecutive tape blocks to disk, starting at Block n1 for n2 blocks.

TIME CL,NO,,60/79,NE
Outputs the system time in hours, minutes, seconds and tenths of seconds.

TINDEX CL,NO,,60/79,NE,"t-index",(n1 n2 ->)
Outputs index listing with title, date, time and page number. Resets TPAGE to 1 at start (+TINDEX does not).

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

TITLE CL,NO,,60/79,NE
To change the title on TLISTS and TINDEX listings, enter TITLE followed by one blank and up to 42 characters of title information terminated by a carriage return or end of screen.

THEN FO,ST,IM&CO,16/21,UN
Used in a colon-definition in the form:
IF ... ELSE ... THEN or
IF ... THEN
THEN marks the point where execution resumes after ELSE (or IF when no ELSE is present).

TL FO,NO,,35/53,CH,"t-l",(n1 n2 ->)
Outputs with line numbers, lines n1 through n2 of the screen whose number is in SCR.

TLINE CL,NO,,60/79,NE,"t-line"
Outputs the title, date, time, and page number line for TLISTS and TINDEX.

TLISTS CL,NO,,60/79,NE,"t-lists",(n1 n2 ->)
Does BLIST starting at Block n1 for n2 blocks, adding title, date, time and page number, formatted three blocks per page. Resets TPAGE to 1 at the start (+TLISTS does not).

TO-B/W FO,NO,,50,NE,"to-b-slash-w",(->) (IBM PC)
Sets current-video-board to black-and-white and does not initialize black-and-white board. (B/W&COLOR option.)

TO-COLOR FO,NO,,50,NE,(->) (IBM PC)
Sets current-video-board to color and does not initialize color board. (B/W&COLOR option.)

TOFF FO,NO,,56/75,UN,"tape-off"
Deactivates the tape recorder motor relay.

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

TOKEN FO,NO,,0,UN,(char addr1 -> addr2)
Starting at addr1, compares each character to char. If char is 32 (ASCII blank), skips any characters equal to char until it encounters a not-equal character. Then counts all not-equal characters until it encounters another equal character or a zero (end-of-data indicator). The found token (string of not-equal characters) is left at HERE (top of dictionary) with the first character equal to the number of characters in the token. The address of the character ending the token (next after end of token) is left as addr2, or zero if no token is found.
This word is used in the parsing routines that set up new words in the dictionary.

TON FO,NO,,56/75,UN,"tape-on"
Activates tape recorder motor relay.

TPAGE CL,NO,UV,60/79,NE,"t-page",(-> addr)
Leaves address of variable holding current page number for TLIST, TINDEX, etc.

TRY TO,NO,,61/80,NE
Diagnostic and learning tool, used in the form:
TRY <name>
Prints the stack contents before and after executing <name>.

TYPE FO,ST,,0,UN,(addr n ->)
Outputs n characters beginning at addr.

U* FO,ST,,0,CH,"u-times",(un1 un2 -> ud)
Performs an unsigned multiplication of un1 by un2, leaving the double number product ud. All values are unsigned.

U. FO,ST,,32/47,NE,"u-dot",(un ->)
Output un, converted according to BASE, as an unsigned number with one trailing blank.

U.R FO,NO,,32/47,NI,"u-dot-r",(un1 n2 ->)
Output un1 as an unsigned number right justified in a field n2 characters wide. Outputs all characters of n1 even when it is wider than n2.

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

U/MOD FO,ST,,0,NE,"u-divide-mod",(ud un1 -> un2 un3)
Perform the unsigned division of double number ud by un1, leaving the remainder un2, and quotient un3. All values are unsigned.

U< FO,ST,,0,NE,"u-less-than",(un1 un2 -> flag)
Leave the flag representing the magnitude comparison of un1 < un2 where un1 and un2 are treated as 16-bit unsigned integers.

UNLN-CUR FO,NO,,38,NE,"underline-cursor",(->) (IBM PC)
Sets the underline cursor character.

UNTIL FO,ST,IM&CO,16/21,CH,(flag ->)
Within a colon-definition, mark the end of BEGIN ... UNTIL loop, which will terminate based on a flag. If flag is true, the loop is terminated. If flag is false execution returns to the first word after BEGIN.
BEGIN ... UNTIL structures may be nested.

UPDATE FO,ST,,0,UN
Marks the most recently referenced block as modified. The block will subsequently be automatically transferred to mass storage should its memory buffer be needed for storage of a different block, or upon execution of SAVE-BUFFERS or FLUSH.

UT FO,NO,UV,0,NE,"u-t",(-> addr)
Leaves the address of a user variable containing the number of blocks to load. This is set to 1 for LOAD, or the number of blocks for LOADS. If executing a LOADS and --> is encountered, UT is reset to 1.

VAL ST,NO,,52/71,UN,($ -> n)
Returns the value n represented by the characters in $ converted according to BASE.

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

VARIABLE FO,ST,,17/22,CH
A defining word executed in the form:
VARIABLE <name>
to create a dictionary entry for <name> and allot two bytes for storage in the parameter field. The application must initialize the stored value. (However, MMSFORTH clears the value to zero.)
When <name> is later executed, it will place the storage address on the stack.

VOCABULARY FO,ST,,17/22,UN
A defining word executed in the form:
VOCABULARY <name>
to create (in the CURRENT vocabulary) a dictionary entry for <name>, which specifies a new ordered list of word definitions. Subsequent execution of <name> will make it the CONTEXT vocabulary. When <name> becomes the CURRENT vocabulary (see DEFINITIONS), new definitions will be created in that list. New vocabularies 'chain' to FORTH. That is, when a dictionary search through a vocabulary is exhausted, FORTH will be searched.

W/0 FO,NO,,48,NE,"w-slash-zero",(->) (IBM PC)
Returns to standard display window.

WBLK FO,NO,,0,UN,"write-block",(addr n ->)
Virtual write-block routine.
Writes Block n from memory at addr. Uses the routine whose code field address (CFA) is stored at 21 MMS 8 + (See Appendix A12 for an explanation of these tables.)

WHILE FO,ST,IN&CO,16/22,CH,(flag ->)
Used in a colon-definition in the form:
BEGIN ... flag WHILE ... REPEAT
Selects conditional execution based on flag. On a true flag, continues execution through REPEAT, which then returns back to just after BEGIN. On a false flag, skips execution to just after REPEAT, exiting the structure.

Attribute(s): IMMEDIATE, COLON-DEFINITION, and/or USER VARIABLE;
Block# (TRS-80/IBM, or 0 if source code not given);
Status: NEW, CHANGED, UNCHANGED (from MMSFORTH V.1.9)

WORD FO,ST,,0,CH,(char -> addr)
Receives characters from the input stream, ignoring leading delimiters if char is ASCII blank (decimal 32) until the non-zero delimiting character, char, is encountered or the input stream is exhausted. The characters are stored as a string with the character count in the first character position. The actual delimiter encountered (char or null) is stored at the end of the text but not included in the count. If the input stream was exhausted as WORD is called, then a zero length will result. The address of the beginning of the string is left on the stack. (In MMSFORTH the address is HERE).

XOR FO,ST,,0,UN,"x-or",(n1 n2 -> n3)
Leave the bitwise exclusive-OR of two numbers.

Y/N FO,NO,,21/38,UN,"y-slash-n",(-> flag)
Prints "(Y/N) ? ", then waits until the Y or N key is pressed. Y puts a zero on the stack, N puts a 1 on the stack. The Y or N is not output. No other characters are accepted as input.

```
: DO-AGAIN  BEGIN CR ." Do again" Y/N UNTIL ;
```

[FO,ST,IM,16/21,NE,"left-bracket"
End the compilation mode. The text from the input stream is subsequently executed. See] .

[COMPILE] FO,ST,IM&CO,17/22,NE,"bracket-compile"
Used in a colon-definition in the form:

```
[COMPILE] <name>
```

Force compilation of the folowing word. This allows compilation of an IMMEDIATE word when it would otherwise be executed.

] FO,ST,,16/21,NE,"right-bracket"
Set the compilation mode. The text from the input stream is subsequently compiled. See [.

Vocabulary or Extension: FORTH, ASSEMBLER, EDITOR, DBL-PREC, STRINGS, CASSETTES, GRAPHICS, RANDOM, ARRAYS, TOOLKIT, CLOCK; 79-STANDARD Status: STANDARD, EXTENSION, NOT-STD;

A10.0 MMSFORTH 8080 ASSEMBLER

A10.1 GENERAL INFORMATION

The MMSFORTH System comes equipped with a full 8080 Assembler which uses about 1.2K of user RAM and provides all necessary instructions for normal work. (Except block moves, which you can do with the MMSFORTH word, CMOVE.) Assembly language is a different subject than Forth,and a challenging one. Instruction books and courses on assemblers are available elsewhere. This section only treats those aspects of our Assembler which differ from more conventional ones.

Those MMSFORTH users who prefer the bulkier (about 3.5K RAM) and less consistent Z80 instruction set, may purchase a full Z80 Assembler as part of the MMSFORTH UTILITIES DISKETTE. Alternatively, individual Z80 instructions may be added to the 8080 Assembler, as can any new words which are hardware compatible. Or, Z80 mnemonics may be defined as synonym definitions. The Assembler cross-reference tables in this appendix are provided to assist you. MMS recommends that you try the 8080 Assembler before settling for one of these less direct and less computer-efficient alternatives.

Both MMSFORTH Assemblers remain unchanged from Version 1.9, with the sole exception that R[(R-Uparrow) is now RP . Also note that ASSEMBLER words are not addressed by the 79-STANDARD.

In general, the 8080 Assembler supplied with MMSFORTH follows the mnemonics of standard 8080 Assembler coding with the word order conventions of the Forth language. This means operands must be entered before the operation code.

Differences occur in the handling of conditional jumps, returns and calls, and in the implementation of the Forth-like functions IF ... ELSE ... THEN, BEGIN ... UNTIL, and BEGIN ... WHILE ... REPEAT.

Conditional operators are used to trigger conditional jumps, calls and returns.

Example:

```
4000  #0 JMPC  is the equivalent of JNZ 4000.
```

Conditional operators are also used with the IF ... ELSE ... THEN, BEGIN ... UNTIL, and BEGIN ... WHILE ... REPEAT constructs in the Assembler in the same way as in Forth.

Example:

```
CY IF CODE1 ELSE CODE2 THEN
```

executes CODE1 if carry flag was set, otherwise executes CODE2.

The structures are:

BEGIN code conditional operator UNTIL
Conditional operator IF code ELSE code THEN
BEGIN code conditional operator WHILE code REPEAT

General use of the MMSFORTH Assembler is illustrated in the following example demonstrating the code for the MMSFORTH word + :

\+ adds the top 2 numbers on the stack.

The code definition is:

```
CODE + HL POP  DE POP  DE DAD  PSH
```

CODE	puts MMSFORTH into ASSEMBLER mode.
+	is name of word being defined.
AX POP	Top of stack is popped into AX
DX POP	2nd on stack is popped into DX
DX AX ADD	DX is added to AX
PSH	AX is pushed back on the stack and control is returned to the Forth Inner Interpreter.

NOTICE THE FOLLOWING:

Use of POPs to get arguments. (In this implementation of Forth the arguments are on the hardware stack.)

Use of PSH to save the result on the stack and return to MMSFORTH. (All code definitions must end with NEXT , PSH , or PSH2.)

Register pair BC, which contains Forth's pseudo program counter, has not been changed.

Register pairs must be addressed by their combined names.

Assembler code is not used within Forth colon definitions.

A10.2 LINKAGE CONVENTIONS USED BY MMSFORTH

BC contains Forth's pseudo program counter.

DE contains the parameter address of the routine invoking this code procedure.

IY contains the address of NEXT.

Arguments from the Forth program are on the stack.

The Version 2.0 Disk System is run with interrupts enabled (they were disabled on Version 1.9).

********** NEVER CHANGE REGISTERS BC **********
(unless you really know what you are doing).

In general it is best to think of the Assembler as not using the stack because, although it does use the stack while generating code, it does not affect the stack during execution of the generated code except explicitly through the use of POP, PUSH, PSH, and PSH2.

A10.3 MMSFORTH ASSEMBLER CROSS-REFERENCE TABLES

The following table is listed in ascending alphabetical order of the standard Z80 Assembler mnemonics. The first column gives the standard Z80 Assembler mnemonic, the second column gives the MMSFORTH Z80 Assembler mnemonic. The third column gives the corresponding MMSFORTH 8080 Assembler mnemonic. The fourth column gives the reference page in the Radio Shack TRS-80 Editor/Assembler Manual for the standard Z80 mnemonic in Column 1.

If you also own THE DATAHANDLER (database management system in MMSFORTH), you are in luck! One of its sample files, named ASSEM, is the source for this table. It can be used quickly and easily to generate print-outs or video displays of selected groupings of words of your choice.

A10.3.1 CROSS REFERENCE LISTING OF 8080 AND Z80 MNEMONICS

Standard-Z80	FORTH-Z-80	FORTH-8080	R.S.Manual
ADC A,S	S A ADC	S ADC	46
ADC HL,SS	SS HL ADC		63
ADD A,(HL)	(HL) A ADD	M ADD	44
ADD A,(IX+D)	D (IX) A ADD		44
ADD A,(IY+D)	D (IY) A ADD		45
ADD A,N	N A ADD	N ADI	43
ADD A,R	R A ADD	R ADD	43
ADD HL,SS	SS HL ADD	SS DAD	63
ADD IX,PP	PP IX ADD		64
ADD IY,RR	RR IY ADD		65
AND N	N AND	N ANI	49
AND S	S AND	S ANA	49
BIT B,R	R B BIT		81
BIT B,(HL)	(HL) B BIT		81
BIT B,(IX+D)	D (IX) B BIT		82
BIT B,(IY+D)	D (IY) B BIT		82
CALL CC,NN	NN CC CALL	NN CC CALLC	93
CALL NN	NN CALL	NN CALL	92
CCF	CCF	CMC	58
CP S	S CP	S CMP	52
CPD	CPD		42
CPDR	CPDR		42
CPI	CPI		41
CPIR	CPIR		41
CPL	CPL	CMA	57
DAA	DAA	DAA	56
DEC IX	IX DEC		67
DEC IY	IY DEC		68
DEC M	M DEC	M DCR	55
DEC SS	SS DEC	SS DCX	67
DI	DI	DI	60
DJNZ E	NN DJNZ		91
EI	EI	EI	60
EX (SP),HL	HL (SP) EX	XTHL	35
EX (SP),IX	IX (SP) EX		36
EX (SP),IY	IY (SP) EX		36
EX AF,AF'	AF' AF EX		34
EX DE,HL	HL DE EX	XCHG	34
EXX	EXX		35
HALT	HALT	HLT	59

Standard-Z80	FORTH-Z-80	FORTH-8080	R.S.Manual
IM 0	0 IM		61
IM 1	1 IM		61
IM 2	2 IM		62
IN A,(N)	N A IN	N IN	98
IN R,(C)	(C) R IN		98
INC (HL)	(HL) INC	M INR	53
INC (IX+D)	D (IX) INC		54
INC (IY+D)	D (IY) INC		54
INC IX	IX INC		66
INC IY	IY INC		66
INC R	R INC	R INR	53
INC SS	SS INC	SS INX	65
IND	IND		101
INDR	INDR		102
INI	INI		99
INIR	INIR		100
JP (HL)	(HL) JP	PCHL	89
JP (IX)	(IX) JP		90
JP (IY)	(IY) JP		90
JP CC,NN	NN CC JP	NN CC JMPC	86
JP NN	NN JP	NN JMP	86
JR C,E	NN C JR		87
JR E	NN JR		87
JR NC,E	NN NC JR		88
JR NZ,E	NN NZ JR		89
JR Z,E	NN Z JR		88

Standard-Z80	FORTH-Z-80	FORTH-8080	R.S.Manual
LD (BC),A	A (BC) LD	BC STAX	20
LD (DE),A	A (DE) LD	DE STAX	20
LD (HL),N	N (HL) LD	N M MVI	17
LD (HL),R	R (HL) LD	R M MOV	15
LD (IX+D),N	N D (IX) LD		17
LD (IX+D),R	R D (IX) LD		16
LD (IY+D),N	N D (IY) LD		18
LD (IY+D),R	R D (IY) LD		16
LD (NN),A	A NN () LD	NN STA	21
LD (NN),DD	DD NN () LD		28
LD (NN),HL	HL NN () LD	NN SHLD	27
LD (NN),IX	IX NN () LD		28
LD (NN),IY	IY NN () LD		29
LD A,(BC)	(BC) A LD	BC LDAX	18
LD A,(DE)	(DE) A LD	DE LDAX	19
LD A,(NN)	NN () A LD	NN LDA	19
LD A,I	I A LD		21
LD A,R	R A LD		22
LD DD,(NN)	NN () DD LD		26
LD DD,NN	NN DD LD	NN DD LXI	24
LD HL,(NN)	NN () HL LD	NN LHLD	25
LD I,A	A I LD		22
LD IX,(NN)	NN () IX LD		26
LD IX,NN	NN IX LD		24
LD IY,(NN)	NN () IY LD		27
LD IY,NN	NN IY LD		25
LD R,(HL)	(HL) R LD	M R MOV	14
LD R,(IX+D)	D (IX) R LD		14
LD R,(IY+D)	D (IY) R LD		15
LD R,A	A R LD		23
LD R,N	N R LD	N R MVI	13
LD R,R'	R' R LD	R' R MOV	13
LD SP,HL	HL SP LD	SPHL	29
LD SP,IX	IX SP LD		30
LD SP,IY	IY SP LD		30
LDD	LDD		39
LDDR	LDDR		40
LDI	LDI		37
LDIR	LDIR		38

Standard-Z80	FORTH-Z-80	FORTH-8080	R.S.Manual
NEG	NEG		57
NOP	NOP	NOP	59
OR N	N OR	N ORA	50
OR S	S OR	S ORA	50
OTDR	OTDR		107
OTIR	OTIR		105
OUT (C),R	R (C) OUT		103
OUT (N),A	A N OUT	N OUT	103
OUTD	OTD		106
OUTI	OTI		104
POP IX	IX POP		33
POP IY	IY POP		33
POP QQ	QQ POP	QQ POP	32
PUSH IX	IX PUSH		31
PUSH IY	IY PUSH		32
PUSH QQ	QQ PUSH	QQ PUSH	31
RES B,M	M B RES		85
RET	RET	RET	94
RET CC	CC RET	CC RETC	95
RETI	IRET		96
RETN	NRET		96
RL M	M RL		73
RLA	RLA	RAL	69
RLC (HL)	(HL) RLC		71
RLC (IX+D)	D (IX) RLC		72
RLC R	R RLC		71
RLCA	RLCA	RLC	69
RLD	RLD		79
RR M	M RR		75
RRA	RRA	RAR	70
RRC M	M RRC		74
RRCA	RRCA	RRC	70
RRD	RRD		80
RST P	P RST	P RST	97
SBC A,S	S A SBC	S SBB	48
SBC HL,SS	SS HL SBC		64
SCF	SCF	STC	58
SET B,(HL)	(HL) B SET		83
SET B,(IX+D)	D (IX) B SET		84
SET B,(IY+D)	D (IY) B SET		84
SET B,R	R B SETT		83
SLA M	M SLA		76
SRA M	M SRA		77
SRL M	M SRL		78
SUB S	S SUB	S SUB	47
XOR N	N XOR	N XRA	51
XOR S	S XOR	S XRA	51

A10.4 **PARTIAL GLOSSARY OF 8080 ASSEMBLER WORDS** (SEE A10.5)

#0 ASSEMBLER Screen 18
NOT EQUAL ZERO conditional operator.

<0 ASSEMBLER Screen 18
LESS THAN ZERO (8080-minus) conditional operator.

=0 ASSEMBLER Screen 18
EQUAL ZERO conditional operator.

>=0 ASSEMBLER Screen 18
GREATER THAN OR EQUAL TO ZERO (8080-positive) conditional operator.

?ARGERR ASSEMBLER Screen 19
Error message generator.

BC ASSEMBLER Screen 18
Registers B and C together.

BEGIN ASSEMBLER Screen 20
Part of the BEGIN ... UNTIL and
BEGIN ... WHILE ... REPEAT structures. Similar to BEGIN in FORTH vocabulary. See MMSFORTH Glossary and explanation of MMSFORTH ASSEMBLER attached to this glossary.

CALL ASSEMBLER Screen 18
8080 op-code, unconditional call.

CALLC ASSEMBLER Screen 19
CALL CONDITIONAL. This plus the conditional operators replace all conditional 8080 calls. See the explanation of MMSFORTH ASSEMBLER attached to this glossary for a fuller description of how conditional calls work.

CY ASSEMBLER Screen 18
CARRY conditional operator.

DE ASSEMBLER Screen 18
Registers D and E together.

ELSE ASSEMBLER Screen 20
Part of the IF ... ELSE ... THEN structure. Similar to ELSE in FORTH vocabulary. See MMSFORTH Glossary and explanation of MMSFORTH ASSEMBLER attached to this glossary.

HL ASSEMBLER Screen 18
Registers H and L together.

IF ASSEMBLER Screen 20
Part of the IF ... ELSE ... THEN structure. Similar to IF in FORTH vocabulary. See MMSFORTH Glossary and explanation of MMSFORTH ASSEMBLER attached to this glossary.

JMP ASSEMBLER Screen 18
8080 op-code, unconditional JUMP.

JMPC ASSEMBLER Screen 19
JUMP CONDITIONAL. This plus the conditional operators replace all conditional 8080 jumps. See the explanation of MMSFORTH ASSEMBLER attached to this glossary for a fuller description of how conditional jumps work.

M ASSEMBLER Screen 18
Memory indicator for 8080 code.

MOV ASSEMBLER Screen 19
8080 op-code.
NOTE: The format for using MOV is:
source-register destination-register MOV
Example:
A B MOV
copies contents of Register A into Register B.

NC ASSEMBLER Screen 18
NO CARRY conditional operator.

NEXT ASSEMBLER Screen 20
Returns to the FORTH inner interpreter.

PE ASSEMBLER Screen 18
PARITY EQUAL conditional operator.

PO ASSEMBLER Screen 18
PARITY ODD conditional operator.

POP ASSEMBLER Screen 19
8080 op-code.

PSH ASSEMBLER Screen 20
Pushes HL, then returns to the FORTH inner interpreter.

PSH2 ASSEMBLER Screen 20
Pushes DE, then HL, then returns to the FORTH inner interpreter.

PSW ASSEMBLER Screen 18
PROCESSOR STATUS WORD (i.e., Register A plus flags).

PUSH ASSEMBLER Screen 19
8080 op-code.

REPEAT ASSEMBLER Screen 20
Part of the BEGIN ... WHILE ... REPEAT conditional structure. Similar to REPEAT in FORTH. See MMSFORTH Glossary and explanation of MMSFORTH ASSEMBLER attached to this glossary.

RET ASSEMBLER Screen 18
8080 op-code.

RETC ASSEMBLER Screen 19
RETURN CONDITIONAL. This plus the conditional operators replace all conditional 8080 returns. See the explanation of MMSFORTH ASSEMBLER attached to this glossary for a fuller description of how conditional returns work.

RP ASSEMBLER Screen 20
Address of the return stack pointer.

SP ASSEMBLER Screen 18
STACK POINTER.

THEN ASSEMBLER Screen 20
Part of the IF ... ELSE ... THEN structure Similar to THEN in FORTH vocabulary. See MMSFORTH Glossary and explanation of MMSFORTH ASSEMBLER attached to this glossary.

UNTIL ASSEMBLER Screen 20
Part of the BEGIN ... UNTIL structure. Similar to UNTIL in FORTH vocabulary. See MMSFORTH Glossary and explanation of MMSFORTH ASSEMBLER attached to this glossary.

WHILE ASSEMBLER Screen 20
Part of the BEGIN ... WHILE ... REPEAT conditional structure. Similar to WHILE in FORTH. See MMSFORTH Glossary and explanation of MMSFORTH ASSEMBLER attached to this glossary.

A10.5 ALL 8080 ASSEMBLER WORDS BY CATEGORY

Return words:
PSH2 PSH NEXT

Conditional jump words:
CALLC JMPC RETC

Conditional operators:
#0 <0 =0 >=0 CY NC PE PO

Conditional structure words:
IF ELSE THEN
BEGIN UNTIL
BEGIN WHILE REPEAT

Registers:
A B BC C D DE E H
HL L M PSW SP

Op-codes:
ACI ADC ADD ADI ANA ANI BCO CALL
CMA CMC CMP CPI DAA DAD DCR DCX
DI EI HLT IN INR INX JMP LDA
LDAX LHLD LXI MOV MVI NOP ORA ORI
OUT PCHL POP PUSH RAL RAR RET RLC
RRC RST SBB SBI SHLD SPHL STA STAX
STC SUB SUI XCHG XRA XRI XTHL

Defining words (to create Assembler):
?ARGERR 1BY 1RG 1Y1 1YP 1YS 2BY
2RG 3BY 3YC CRG

Special:
RP

A11.1 **TRS-80 KEYBOARD KEY DEFINITIONS** (MMSFORTH ASCII codes)

KEY-SYMBOL			KEY (Dec/Hex)		Shift-KEY (Dec/Hex)		Cntrl-KEY (Dec/Hex)		Alternate-KEY (Dec/Hex)	
@	`		64	40	96	60	X	X	128	80
A	a		65	41	97	61	1	1	129	81
B	b		66	42	98	62	2	2	130	82
C	c		67	43	99	63	3	3	131	83
D	d		68	44	100	64	4	4	132	84
E	e		69	45	101	65	5	5	133	85
F	f		70	46	102	66	6	6	134	86
G	g		71	47	103	67	7	7	135	87
H	h		72	48	104	68	8	8	136	88
I	i		73	49	105	69	9	9	137	89
J	j		74	4A	106	6A	10	A	138	8A
K	k		75	4B	107	6B	11	B	139	8B
L	l		76	4C	108	6C	12	C	140	8C
M	m		77	4D	109	6D	13	D	141	8D
N	n		78	4E	110	6E	14	E	142	8E
O	o		79	4F	111	6F	15	F	143	8F
P	p		80	50	112	70	16	10	144	90
Q	q		81	51	113	71	17	11	145	91
R	r		82	52	114	72	18	12	146	92
S	s		83	53	115	73	19	13	147	93
T	t		84	54	116	74	20	14	148	94
U	u		85	55	117	75	21	15	149	95
V	v		86	56	118	76	22	16	150	96
W	w		87	57	119	77	23	17	151	97
X	x		88	58	120	78	24	18	152	98
Y	y		89	59	121	79	25	19	153	99
Z	z		90	5A	122	7A	26	1A	154	9A
1	!	|	49	31	33	21	124	7C	X	X
2	"	^	50	32	34	22	94	5E	X	X
3	#		51	33	35	23	159	9F	X	X
4	$		52	34	36	24	31	1F	X	X
5	%		53	35	37	25	X	X	X	X
6	&		54	36	38	26	X	X	X	X
7	'	`	55	37	39	27	96	60	X	X
8	(	[	56	38	40	28	91	5B	X	X
9	)	]	57	39	41	29	93	5D	X	X
0			48	30	X	X	128	80	X	X
:	*	~	58	3A	42	2A	126	7E	X	X
;	+		59	3B	43	2B	127	7F	X	X
,	<	{	44	2C	60	3C	123	7B	X	X
-	=	_	45	2D	61	3D	95	5F	X	X
.	>	}	46	2E	62	3E	125	7D	X	X
/	?	\	47	2F	63	3F	92	5C	X	X

KEY-SYMBOL	KEY (Dec/Hex)		Shift-KEY (Dec/Hex)		Cntrl-KEY (Dec/Hex)		Alternate-KEY (Dec/Hex)	
Enter	13	0D	13	0D	13	0D	13	0D
Space	32	20	32	20	32	20	32	20
Uparrow	27	1B	155	9B	187	BB	219	DB
Downarrow	28	1C	156	9C	188	BC	220	DC
Leftarrow	29	1D	157	9D	189	BD	221	DD
Rightarrow	30	1E	158	9E	190	BE	222	DE

Notes:
X = no character generated.
Model I display characters may not correspond to above.

A11.2 MMSFORTH CRT CONTROL CODES (TRS-80)

VALUE (Dec/Hex)		VIDEO EFFECT (When EMITted)
08	08	Backspace cursor and erase character.
09	09	Tab (0,8,16,...)
10	0A	LF - move cursor to start of next line and erase line.
12	0C	Page - move cursor to upper left corner and erase screen.
13	0D	CR - move cursor to start of next line and erase line.
14	0E	Turn cursor on.
15	0F	Turn cursor off.
21	15	Swap space compression/special characters.
22	16	Swap special/alternate characters.
23	17	Double width characters.
24	18	Backspace cursor one character.
25	19	Advance cursor one character.
26	1A	Move cursor down one line.
27	1B	Move cursor up one line.
28	1C	Move cursor to upper left corner of screen.
29	1D	Move cursor to start of line.
30	1E	Erase to end of line.
31	1F	Erase to end of screen.

A12.0 **SYSTEM CONSTANTS** (MMSFORTH V2.0, TRS-80 DISK SYSTEM)

The following tables are included to assist advanced users in their own experiments. Like most internal details, MMS does **not** support them as a part of the system license but will provide such support as a consulting service.

DISK DATA:

n DISKDATA	Array of 8 groups, each with 9 bytes for 1 drive, plus 2 bytes. The group after the highest drive defined contains zero in the first two bytes.
0 + @	#blocks on drive.
2 + @	#bytes per sector.
4 + C@	Step speed code byte.
5 + C@	High track number.
6 + C@	First sector number on track.
7 + C@	Last sector number-plus-one on track.
8 + C@	Bits 0..3 - drive select bit, MODEL I & III Bit 4 (16) - side 0/1, MODEL III only Bit 7 (128) - single/double density MODEL III only

BUFFER DATA:

n BUFFDATA	Array of 2 to n groups, of 6 bytes for each block buffer.
0 + @	Block# currently in buffer; -1 = empty.
2 + C@	Blocks usage sequence#.
3 + C@	Update byte.
4 + @	Addr of buffer, buffer is followed by two bytes of zeroes.

CURSOR DATA:

CURSOR @	Character for cursor display.
2 + C@	Cursor on-time cycle count. System default=9.
3 + C@	Cursor blink cycle count. System default=24.
4 + C@	Minor timing.
5 + C@	Major timing. Cursor speed scaler = minor timing + 256*major timing (if minor=0, it's really 256).
6 + C@	0=normal uppercase, 32=normal lowercase keyboard.

MMS(FORTH) DATA:

n MMS — Array of system addresses.
If n is:

0	Addr of Inner Interpreter (NEXT).
1	Code Addr for Vocabularies.
2	Code Field Addr (CFA) of JMP.
3	CFA of JMP conditional (0 TOS causes JMP).
4	Addr of 16-bit comparison subroutine (HL minus DE).
5	Addr of 32/16-bit subroutine (HLDE pair div by BC giving HL quotient and DE remainder.
6	Addr of 2's-complement subroutine (HL = work reg.).
7	Addr of double-2's-complement subr. (HLDE pair).
8	Addr of 16*16-bit subroutine (unsigned) (HL times DE giving HLDE pair).
9	CFA of :
10	CFA of ;
11	Addr of ORIGIN sub-table.
12	Addr of DISK ROUTINE sub-table.
13	Addr of INITIALIZE sub-table.
14	Addr of REAL-TIME CLOCK sub-table.
15	Addr of disk start-up delay byte.
16	CFA of print string literal.
17	CFA of byte literal.
18	CFA of double word literal.
19	CFA of ;CODE.
20	Addr of DEFAULT SYSTEM VALUES sub-table.
21	Addr of CURRENT SYSTEM VALUES sub-table.
22	CFA of word literal.
23	CFA of routine which looks up a hash-coded name (on the stack as a double number) and returns CFA of name or zero.
24	CFA of hash-encoding routine (takes string addr of name and leaves double number encoded name).

ORIGIN:

11 MMS	First memory location of Forth code and entry point to MMSFORTH (on standard TRS-80 is 19200D, 4B00H). Contains jump to system initialization.
3 + @	Address of end of memory plus one.
5 + @	Negative size of Return Stack.
7 + @	Parameter Field Address (PFA) of Outer Interpreter word which is executed at ABORT or QUIT.

DISK ROUTINE:

12 MMS	Address of DISK ROUTINE values.
0 + C@	Early-Disable-Interrupt byte.
2 + C@	Disk number-of-tries-before-reject byte.

INITIALIZE:

13 MMS	INITIALIZE TABLE of five addresses that are scanned during ABORT/QUIT processing. For each: =0 do nothing, #0 execute subroutine at that address.
0 + @	Unused.
2 + @	Type-ahead.
4 + @	Future spooler.
6 + @	Unused.
8 + @	Unused.

REAL-TIME CLOCK:

14 MMS	REAL-TIME CLOCK INTERRUPT TABLE of five addresses that are scanned at each real-time interrupt. For each: =0 do nothing, #0 executed subroutine at that address.
0 + @	Model I Real-time Clock (Unused on Model III).
2 + @	Type-ahead.
4 + @	Future Spooler.
6 + @	Unused.
8 + @	Unused.

DEFAULT SYSTEM VALUES:

20 MMS — DEFAULT SYSTEM VALUES TABLE. When source is recompiled, these values are copied into the corresponding entries of 21 MMS.

0 + @	Entry address of Full ASCII keyboard subroutine.
2 + @	Entry address of Blinking Cursor keyboard subroutine.
4 + @	Entry address of TRS-80 display output calling subroutine.
6 + @	CFA of Disk Block-Read word.
8 + @	CFA of Disk Block-Write word.
10 + @	CFA of word to output _ok & CR (_ok: if compile state).
12 + @	CFA of 79-STANDARD EXPECT word.
14 + @	CFA of null word (default extend-interpreter word).
17 + @	PFA of ABORT (default Break definition).
20 + @	PFA of %CONT (or Screen-Print word when loaded).

CURRENT SYSTEM VALUES:

31 MMS — CURRENT SYSTEM VALUES TABLE. May be changed by the careful user.

0 + @	Address of ?KEY driver subroutine (value in A). ?IN
2 + @	Address of KEY driver subrtn (value in A). IN-UNIT
4 + @	Address of EMIT driver subrtn (value in A). OUT-UNIT
6 + @	CFA of Mass Storage Read routine (ERBLK).
8 + @	CFA of Mass Storage Write routine (EWBLK).
10 + @	CFA of prompt routine.
12 + @	CFA of EXPECT execution word.
14 + @	CFA of Extend-interpreter word (executed between dictionary lookup and number conversion).
17 + @	PFA of Break definition (must end in ABORT or %CONT).
20 + @	PFA of C-* (Screen-Print) definition (must end in ABORT or %CONT).
22 + @	Amount of memory to leave between top of dictionary and stack. When actual memory available gets down to this size a warning message is given.
24 + C@	Auto-Boot flag. =0 - no automatic execution after copyright message. #0 - after copyright message, execute Auto-command as set with CUSTOMIZE.
25 + C@	Dup-name: flag. =0 - no message when duplicate word name detected. #0 - warning message is printed when duplicate word name detected (default value).

Table 12 - **FORTH PROGRAMMING RULES**

1. FORTH WORDS ARE COMPOSED OF UP TO 31 PRINTABLE CHARACTERS, SEPARATED BY SPACES.

2. MOST WORDS REQUIRE PARAMETERS ON A PUSH-DOWN STACK.

3. THE BREAK KEY OR ANY ERROR MESSAGE EMPTIES BOTH STACKS.

4. ALL PARAMETERS PUT ONTO A STACK MUST BE REMOVED WHEN THEY ARE NO LONGER NEEDED. THE ORDER WILL BE **LAST IN, FIRST OUT.**

5. ALL WORDS MUST BE DEFINED BEFORE THEY CAN BE USED.

6. COMPILING WORDS MUST NEVER BE USED OUTSIDE A DEFINITION.

7. EVERY **IF** MUST BE FOLLOWED BY A **THEN.**

8. ANYTHING PUSHED ONTO THE RETURN STACK MUST BE REMOVED WITHIN THE SAME DEFINITION.

9. WHEN NESTING STRUCTURES IN FORTH, YOU MUST NEST EACH STRUCTURE COMPLETELY WITHIN ANY OUTER STRUCTURE.

10. NEVER PUT AN UNTESTED ROUTINE INTO A LOOP.

11. A STUB MUST REPRODUCE THE BEHAVIOR OF ITS INTENDED COUNTERPART WITH RESPECT TO STACK USAGE.

A14.0 **BIBLIOGRAPHY**

Additional Forth books are available now and more are expected in the near future. Some other types of books are particularly valuable to advanced users of MMSFORTH on the Radio Shack TRS-80 microcomputers. MMS recommends and stocks the following:

MMSFORTH NEWSLETTER — $10.00/year
(V.1, 1980; V.2, 1981; V.3, 1982)
By subscription to licensed MMSFORTH users, only.

STARTING FORTH (L.Brodie) — Hardcover: $19.95; Softcover: $15.95
The best on learning Forth; get it!

MMSFORTH USERS MANUAL — $17.50
Non-user version of this manual (less Appendices).

INTRODUCTION TO FORTH (K.Knecht) — $9.95
On MMSFORTH V1.9; detailed attention for the beginning Forth user who has Extended BASIC experience.

Special FORTH Issue of BYTE MAGAZINE (Aug. 1980) — $4.00
A collector's item for Forth users and beginners.

FORTH-79 STANDARD MANUAL — $13.95
The official description of the 79-STANDARD subset of Forth words.

THREADED INTERPRETIVE LANGUAGES (R.Loeliger) — $18.95
Advanced, excellent analysis of the internals of a MMSFORTH-like language. (How to build one!)

PROGRAM DESIGN AND CONSTRUCTION (D.Higgins) — $13.95
Good introduction to structured programming.

TRS-80 ASSEMBLY LANGUAGE (H.Howe,Jr.) — $9.95
Good overview of Assembler and firmware considerations.

MICROSOFT BASIC DECODED & OTHER MYSTERIES FOR TRS-80 (J.Farvour) — $29.95
Excellent for use of TRS-80 Model I firmware, much of it good for Model III, also.

MODEL III ROM COMMENTED (SSM Inc.) — $22.50
Complete Model III listing to supplement the above.

THE 8086/8088 PRIMER (S.Morse) $11.95
Assembler and related information for the IBM Personal Computer, by one of the designers of its CPU chip.

All plus shipping/handling charges and Massachusetts tax, and subject to change by MMS, publishers, the Postal Service, et al. Present shipping costs are $2.00 minimum plus $1.00 per additional book, or plus 20% for most overseas Air Mail. No unpaid purchase orders, please. To assure adequate payment, use Visa, MasterCard, UPS COD, or include an extra payment; MMS will return any overpayment with your order.